NOVA IMS
Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

SISTEMA DE RECOMENDAÇÕES EM TEMPO REAL COM SPARK-STREAMING

Guilherme Peres Barros

Trabalho de Projeto apresentado como requisito parcial para obtenção do grau de Mestre em Métodos Analíticos Avançados

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

SISTEMA DE RECOMENDAÇÕES EM TEMPO REAL COM SPARK-STREAMING

por

Guilherme Peres Barros

Trabalho de Projeto apresentado como requisito parcial para a obtenção do grau de Mestre em Métodos Analíticos Avançados

Orientador/Coorientador: Mijail Naranjo-Zolotov

Julho 2021

DEDICATÓRIA

Dedico de todo meu coração, à minha filha Daniela.

AGRADECIMENTOS

À minha família, principalmente aos meus pais, Márcio e Marisa, pela educação. Muito obrigado por insinar-me a pescar.

À minha esposa, Tatiana e à minha filha Daniela, pela parceria, pelo carinho e pelo amor, que tornaram a realização deste trabalho possível.

Ao Professor Mijail Naranjo-Zolotov, pelo apoio durante a realização deste trabalho e a todos professores da NOVA IMS e aos colegas por estarem sempre presentes.

RESUMO

Apresenta-se o nome Triangulum como fictício para preservação dos dados da empresa. A empresa Triangulum Comunicação Ltda. é umas das maiores no segmento de comunicação de Portugal, vive o momento que a informação também gera dados e mais dados. Hoje o número de *pageviews* e o tempo de navegação do utilizador nos sites, são dois dos fatores mais importantes para a receita da Triangulum. Cada utilizador tem suas características de navegação, isso é de extrema relevância para a análise dos dados. Os artigos dos sites visitados têm sempre, ou quase sempre, uma referência, que pode ser um artigo, uma rede social ou até um mecanismo de pesquisa. Com essas informações iremos aplicar modelos de *machine learning* com o objetivo de fazer recomendações para os próximos artigos a serem lidos, de forma que o utilizador navegue pelos sites, desfrutando do conteúdo, atingindo maior tempo e mais *pageviews*.

A coleta dos dados foi feita pela Triangulum, disponibilizando assim os dados de forma bruta. Após essa etapa de extrair, carregar e transformar, os dados armazenados no *data lake* são preparados e então partimos para a análise dos dados, será aplicado aos modelos de *machine learning* para a implementação do sistema de recomendação. O resultado é aplicado a cada artigo de todos os sites da Triangulum, ocupando parte do espaço disponível para as recomendações de leitura.

PALAVRAS-CHAVE

Sistema de recomendação; Filtragem colaborativa; Aprendizado de máquina; Kafka; Processamento em tempo real; Hadoop; Spark.

ABSTRACT

The name Triangulum is presented as fictitious to preserve the company's data. The company Triangulum Communication Ltda. is one of the largest in the media segment in Portugal, experiencing the moment that information also generates data and more data. Today the number of pageviews and the user's time spent browsing the sites are so important for the company's revenue. Each user has their own navigation characteristic, this is extremely important for data analysis. The articles of the visited sites always have, or almost always, a reference, that can be an article, a social network or even a search engine. With this information we will apply machine learning models in order to make recommendations for the next articles to be read, so that the user navigates through the sites, enjoying the content, reaching more time and more pageviews.

The data collection was done by the company, thus making the data available in a gross way. After this extract, load, and transform stage, the data stored in the data lake goes through a preparation phase before the data analysis, it will be applied to the machine learning models for the implementation of the recommendation system. The results are applied to each article on all company websites, taking up part of the space available for reading recommendations.

KEYWORDS

Recommendation system; Collaborative filtering; Machine learning; Kafka; Real-time processing; Hadoop; Spark.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

AI	Artificial Intelligence
AM	Application Master
API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration/Continuous Delivery
CPU's	Central Process Unit
EC2	Elastic Compute Cloud
ETL	Extract Transform Load
GB	Gigabyte
HDFS	Hadoop Distributed File System
IP	Internet Protocol
JDK	Java Development Kit
MLlib	Machine Learning Library
MR	MapReduce
PC's	Personal Computer
POCs	Proof of Concept
RAM	Random Access Memory
RDDs	Resilient Distributed Datasets
RM	Resource Manager
RPM	Revolutions Per Minute
S3	Simple Storage Service
SCP	Secure Copy
SEO	Search Engine Optimization
SQL	Structured Query Language
SSH	Secure Shell
SVD	Singular Value Decomposition

URL	Uniform Resource Locator
VPC	Virtual Private Cloud
v's	Volume, Velocity, Variety, Veracity
YARN	Yet Another Resource Negotiator

1. INTRODUÇÃO

A era digital está proporcionando grande dificuldade para as pessoas escolherem entre uma grande variedade de conteúdos e artigos e entre as várias alternativas que lhe são apresentadas. Com isso vem junto o *big data* (*Big Data Analytics*, 2012), caracterizado pelos seus quatros v's, são eles volume, velocidade, variedade, veracidade (Hitzler e Janowicz, 2009); os algoritmos de *machine learling* (Alpaydin, 2020), e surgem também os sistemas de recomendação computacionais (Personalized, 2006). A evolução destes sistemas e o fato deles trabalharem com grandes volumes de informações permitiram que recomendações possam ser atingidas, proporcionando ainda um maior nível de confiança.

Com o acesso aos dados, transformando-os em informação, as pessoas têm o mundo em suas mãos, para obterem conteúdos interessantes, informáticos, trágicos, políticos, populares, saúde, beleza, dentre outros. Passou-se então a ter milhares de dados brutos produzidos a cada instante, a próxima hora já produz mais e mais dados. Hoje, não sabemos se o utilizador do *website* está se interessando pelos próximos artigos a serem lidos. Queremos então produzir uma direção a leitura, para que o utilizador se interesse pelo próximo artigo, e pelo seguinte, assim sucessivamente. Considerando que cada acesso a um artigo vem de uma referência e que características semelhantes de acessos irão ocorrer, quem leu o artigo A também leu artigo B.

Este projeto envolve a informação na forma de leitura digital e a análise dos dados das dezenas dos *websites* do grupo, esses que são produzidos pela visualização dessas informações adquiridas para leitura como notícias, entretenimento, desporto, dentre outros, com milhões de acessos diariamente gerando um grande volume de dados, temos como objetivo principal usar de algoritmos de *machine learning* para criar um sistema de recomendações de artigos relacionados aos históricos de leitura, usufruindo do comportamento de todos utilizadores, de modo que o utilizador atual continue a navegar, gerando assim maior número de acessos diários.

O principal objetivo é de fato produzir recomendações a partir dos históricos de visitas aos *sites*, a precisão não é o ponto em questão, porém, a uma ideia de acompanhamento dos resultados nos próximos meses após a implementação do projeto. Portanto, o utilizador tendo um conteúdo recomendado que o atrai, nós por outro lado, estamos a satisfazer o nosso cliente e como consequência gerar maior receita, pois quanto mais visualizações e acessos obtivermos maiores captações de publicidade ganharemos no futuro.

Este documento está distribuído da seguinte forma: o Capítulo 1 introduz o problema e apresenta os objetivos do trabalho; no Capítulo 2 é feita a revisão da literatura, o que as companhias usam para sistemas de recomendações, principais no ramo de recomendação; a metodologia utilizada é destaque no Capítulo 3 onde são mostradas a origem dos dados, como está armazenado e as variáveis utilizadas, além de explicar o funcionamento do modelo; por fim, no Capítulo 4 em diante, são apresentadas as conclusões, as limitações e propostas para trabalhos futuros.

2. REVISÃO DA LITERATURA

2.1. INTELIGÊNCIA ARTIFICIAL

O conceito de Inteligência Artificial (Entwistle, 2004) que aqui chamaremos de AI (do inglês, *Artificial Intelligence*). A AI é atualmente um "tópico quente": a cobertura da mídias e a discussão pública sobre AI são quase impossíveis de evitar. (Lima, Pinheiro e Santos 2014), que afirmam que "a I.A. é o conjunto de ações que, se fossem realizadas por um ser humano, seriam consideradas inteligentes." É uma definição que nos chama a comparar as ações dos computadores com as nossas próprias para definirmos se estamos diante de um comportamento inteligente ou não. Em que pese ser uma definição útil, não é de todo precisa, uma vez que há algumas ações que o computador consegue realizar que não traduzem adequadamente um comportamento inteligente. Dentre as principais aplicações da IA, destacamos, carros autônomos (Trindade, 2018), recomendação de conteúdo (Meteren e Someren, 2000) e processamento de imagem e vídeo.

Os carros autônomos exigem uma combinação de várias técnicas de AI: busca e planeamento para encontrar a rota mais conveniente de A a B; visão computacional para identificar obstáculos; e tomada de decisão sob incerteza para lidar com o ambiente complexo e dinâmico. Cada um deles deve trabalhar com precisão quase perfeita para evitar acidentes. As mesmas tecnologias também são usadas em outros sistemas autônomos, como robôs de entrega, drones e navios autônomos.

Muitas das informações que encontramos no decorrer de um dia comum são personalizadas, denominadas recomendação de conteúdo. Exemplos incluem Facebook, Twitter, Instagram e outros conteúdos de mídias sociais; anúncios online; recomendações de música no Spotify; recomendações de filmes na Netflix, HBO e outros serviços de *streaming*. Muitos editores *on-line*, como sites de companhias de jornais e de radiodifusão, bem como mecanismos de pesquisa, como o Google, também personalizam o conteúdo que oferecem.

Embora a primeira página da versão impressa de um jornal seja a mesma para todos os leitores, a primeira página da versão *online* pode ser diferente para cada utilizador. O algoritmo é que determinam o conteúdo a ser exibido nas páginas do *website*.

O reconhecimento facial já é uma *commodity* usada em muitos aplicativos, companhias e governos, como por exemplo: a organização de suas fotos de acordo com as pessoas, a marcação automática nas mídias sociais e o controle de passaportes. Técnicas de processamento de imagem e vídeo, podem ser usadas para reconhecer outros carros e obstáculos em torno de um carro autônomo, ou para estimar populações de vida selvagem.

A Figura 1, ilustra o que abrange AI e o que se encaixa dentro da AI.

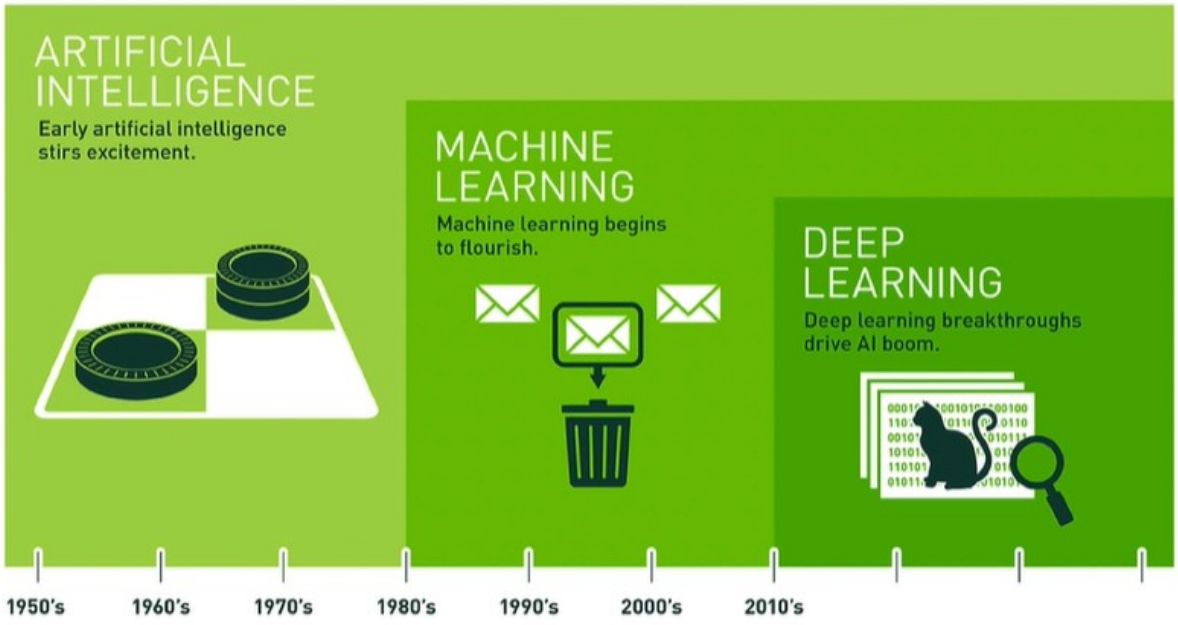

Figura 1 - Inteligência Artificial linha do tempo. (https://semiengineering.com/artificial-intelligence-chips-past-present-and-future/)

2.2. MACHINE LEARNING

Machine learning (Cronin, 2017), onde a tecnologia computacional nos levou, proporcionando a mais perfeita análise dos dados gerando a uma informação valiosa, como se fosse uma regra de três composta passando para o algoritmo uma dezena de variáveis e obtendo uma resposta precisa do "*target*" desconhecido, podendo ser aplicado seja ele a previsão do tempo ou o resultado de um jogo esportivo.

Em 1959, Arthur Samuel definiu *machine learning* como o "campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados" (Weiss, 2002), não é como o *machine learning* (Gao e Jamidar, 2014) de hoje graças a evolução tecnológica e a capacidade de processamento maior e mais rápida. O principal objetivo é aprender com os dados históricos disponíveis, sejam dados de vendas, compras, consumo. O próprio modelo sofre mutações e se adapta assim que novas informações são inseridas na base de dados, contando com o apoio e usufruindo de técnicas da matéria de estatística e ciência da computação para ser mais preciso (*Machine learning: o que é e qual sua importância? | SAS*, 2021).

Quando falamos em resultados e previsões, muitas vezes estamos trabalhando em tempo real, onde a tomada de decisão é crucial para o lucro de uma companhia, podendo assim receber uma nova informação e em questões de milésimos de segundos prever se uma compra com o uso de um cartão de crédito bancário é uma fraude ou não (Sathyapriya e Thiagarasu, 2015). Há diversas áreas de aplicações de algoritmos de *machine learling*, tais como; carros autônomos, recomendações na Amazon e Netflix, combinado com *textmining* para classificar um comportamento de um cliente, imagens de vídeos para detetar situação anormais, ajuda na medicina (Cruz e Wishart, 2006).

E como a máquina aprende? Vamos fazer uma analogia a uma criança, mostramos a ela uma imagem de um cavalo pela primeira vez em sua vida, ela está aprendendo, certo? Vejamos então, está criança nunca viu uma zebra, pela primeira vez que essa criança vai ao zoológico e vê uma zebra, ela vai dizer que é um cavalo, nunca ele tinha visto uma zebra antes, então seus responsáveis vão corrigir e dizer que aquele animal não um cavalo e sim uma zebra, e é assim que as máquinas aprendem, no caso com supervisão.

Vejamos alguns tipos de algoritmo que podem ser utilizados para gerar recomendações.

2.2.1. Collaborative filtering

A Filtragem Colaborativa (Gandhi *et al.*, 2020) (Shinde, 2015) constitui-se em uma das mais populares técnicas de recomendação, sendo utilizada em muitos sistemas existentes na *Internet* (Schafer et al., 2000).

A técnica olha para um grupo de pessoas e analisa suas preferências em comuns. O interesse comum e semelhantes possuídos por experiências entre pessoas em determinados itens, artigos, sites, dentre outros é essência desta técnica, essa troca de informação pode facilitar as recomendações de conteúdos filtrando com base nas avaliações (*feedback*) feitas pelos utilizadores sobre os mesmos itens.

Observar o comportamento de um grupo procurando ter um resultado de uma avaliação positiva ou negativa sobre determinados itens permitindo assim poder analisar as similaridades entre os observados dentro do grupo. Os grupos podem por sua vez ser formados pela sua similaridade obtida nos resultados, tendo assim como identificar vizinhos mais próximos (Adomavicius e Tuzhilin, 2005).

Um processo de Filtragem Colaborativa preliminar é cálculo do coeficiente de similaridade entre dois utilizadores (Cazella et al., 2009), incluindo a seleção dos grupos de vizinhos que são calculados com algoritmos de *clustering* (Lachi e Vieira Da Rocha, 2005).

2.2.1.1. Factoração de matriz

A factoração de matriz (Zhang *et al.*, 2018) é um algoritmo que multiplica dois tipos diferentes de dados. A filtragem colaborativa é a técnica, como vimos no item 2.2.1, que usa a factoração de matriz para identificar a relação entre os itens e os utilizadores, atribuindo assim o peso (*ranting*) para cada combinação. Com a entrada nos artigos dos *websites* pelos utilizadores, cada artigo soma uma visualização, proporcionando uma pontuação para cada artigo, facilitando assim o desenvolvimento do projeto e permitindo o cálculo dos pesos.

Para a parte da recomendação, a única parte que é atendida é a factoração da matriz (Bokde, Girase e Mukhopadhyay, 2015) que é feita a matriz de classificação do item do utilizador. A factoração de matrizes consiste em 2 matrizes cujo produto é a matriz original. Os vetores são usados para representar o item 'q_i' e o utilizador 'p_u', de modo que seu produto escalar seja a classificação esperada.

$$rating = \hat{r}_{ui} = q_i^T p_u \quad (1)$$

O 'q_i' e 'p_u' podem ser calculados de tal forma que a diferença de erro quadrada entre o produto escalar do utilizador e do item e as classificações originais na matriz do utilizador-item seja mínima.

$$\min(p,q) = \sum_{(u,i)\varepsilon k} \left(r_{ui} - q_i^T \cdot p_u \right)^2 \quad (2)$$

Regularização: evita o ajuste excessivo do modelo,um aspeto importante de qualquer modelo de *machine learling*, pois resulta em baixa precisão do modelo. A regularização elimina o risco de *overfitted*.

Para este efeito na regularização, um termo de penalização é introduzido na equação de minimização acima. λ é o fator de regularização que é multiplicado pela soma quadrada das magnitudes dos vetores do utilizador e do item.

$$\min imum(p,q) = \sum_{(u,i)\varepsilon k} \left(r_{ui} - q_i^T \cdot p_u \right)^2 + \lambda \left(\left\| q_i \right\|^2 + \left\| p_u \right\|^2 \right) \quad (3)$$

A decomposição de valores singulares, também conhecida como algoritmo SVD (do inglês, *Singular Value Decomposition*), é usada como método de filtragem colaborativa em sistemas de recomendação (Sharma e Kaur, 2016). SVD é um método de factoração de matriz que é usado para reduzir os recursos nos dados, reduzindo as dimensões de N para K, onde (K<N).

Os modelos de factoração de matriz mapeiam utilizadores e itens para um espaço de fator latente conjunto com dimensionalidade f, as interações utilizador-item são modeladas como produtos internos naquele espaço. Assim, cada item i está associado a um vetor $q_i \in R^f$, e cada utilizador u está associado a um vetor $p_u \in R^f$. Para um determinado item i, os elementos os q_i medem até que ponto o item possui esses fatores positivos ou negativos. Para um determinado utilizador u, os elementos de p_u medem a extensão do interesse que o utilizador tem em itens que são altos nos fatores positivos ou negativos correspondentes. O produto escalar resultante $q_i^T p_u$ captura a interação entre o utilizador u e o item i (*rating*). os utilizadores têm todo o interesse nas características do item, atribuindo o peso 1.0. Isso se aproxima da classificação do utilizador u's do item i, que é denotada por r_{ui} levando à estimativa.

2.2.2. Hybrid recurrent neural network

O hybrid recurrent neural network (Sigtia, Boulanger-Lewandowski e Dixon, 2015) é uma técnica muito bem utilizada pela Amazon, que por sinal é nossa prova de conceito, onde seus produtos são recomendados a partir das recorrências de outros utilizados em seu *website* partilhando da interações de cada clique de um utilizador com o item a ser consumido.

O *recurrent neural network* é um modelo de rede neural proposto nos anos 80 (Rumelhart et ai., 1986; Elman, 1990; Werbos, 1988) para modelagem séries temporais. A estrutura da rede é semelhante à a de um *multilayer perceptron*, com a distinção de permitir conexões entre unidades ocultas associado a um atraso de tempo. Por meio dessas conexões, o modelo pode reter informações sobre o passado, permitindo descobrir correlações temporais entre eventos que estão longe um do outro nos dados (Rosindell e Wong, 2018).

A AWS (do inglês, *Amazon Web Services*) disponibiliza aos seus clientes um serviço que permite utilizar algoritmos semelhentes ao usados em seu *website* amazon.com, o AWS Personalize (https://aws.amazon.com/pt/personalize/). O Personalize consiste em receber informações das transações históricas ocorridas em um determinado item, que são transações entre utilizador e produto, podendo ser completada claramente com informações adicionais dos utilizadores como; idade, gênero, salário, podendo ser as melhores variáveis para o modelo, isso também ocorre com o produto. Segue abaixo o fluxograma do Personalize, Figura 2.

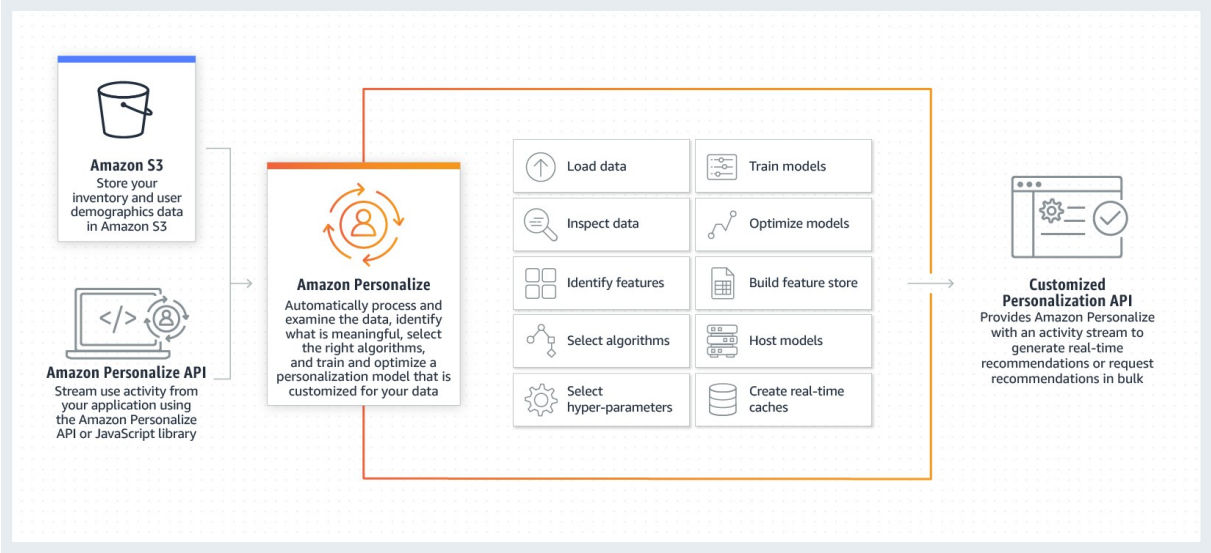

Figura 2 - AWS Personalize esquema. (https://aws.amazon.com/pt/personalize/)

2.3. SISTEMAS DE RECOMENDAÇÕES

Os sistemas de recomendações (Meteren e Someren, 2000) surgem com a necessidade que o mundo digital vive, são inúmeros conteúdos para analisarmos, após uma simples pesquisa nos *sites* de buscas falta tempo para que os utilizadores possam tomar uma decisão de escolha. Com as novas tecnologias e com a inteligência computacional os próprios *sites* de busca usam algoritmos para exibir os *sites*, artigos ou resultados de busca, seja como for, por ordem de preferências, maior visualização e também com um ótimo trabalho de SEO (do inglês, *Search Engine Optimization*) (https://yoast.com/). O resultado da combinação dessas técnicas podem colocar um *site* de pesquisa na primeira página, dado que a segunda página de resultado de busca é considerada como o "*limbo*" (Pasquinelli, 2009).

A evolução dos sistemas de recomendações contempla bibliotecas digitais, *sites* de compras de produtos, *sites* de jornais, dentre outros. Formam feitos diversos projetos de bibliotecas digitais

(Cazella et al., 2009), segundo esse artigo, as pessoas que usufruíram desses projetos tiveram melhor aproveitamento de aprendizagem em um curso, por exemplo, utilizando aprendizado colaborativo, alunos que leram conteúdos A e também leram outros conteúdos B, quem leu B pode muito bem ler conteúdo A e ter melhor aprendizado.

Embora a recomendação personalizada seja uma característica desejável em qualquer sistema computacional, independente da área do conhecimento, não são todas as companhias que conseguem desenvolver seus próprios algoritmos, desde modo tendo que recorrer as companhias específicas no conteúdo abordado como; Taboola (https://www.taboola.com/), Engageya (https://www.engageya.com/), que são exemplos de companhias que utilizam os dados de acessos ao *website* de seus clientes e partir dessa informação geram recomendações utilizando algoritmos com citados nos itens 2.2.1 e 2.2.2 desse capítulo.

2.3.1. O que são Sistemas de Recomendações

Sistemas de recomendação são algoritmos de *machine learning* capazes de analisar o comportamento de utilizadores de uma plataforma de vídeo ou de um *website* e de compreender o padrão dele para proporcionar recomendações relevantes e personalizadas de conteúdos novos. Esses sistemas preveem o produto mais provável que os utilizadores têm maior probabilidade de comprar e se interessar.

Os sistemas de recomendações lidam com um grande volume de informações presentes, filtrando as informações mais importantes com base nos dados fornecidos por um utilizador e outros fatores que atendem à preferência e interesse do utilizador. Ele descobre a correspondência entre utilizador e item e imputa as semelhanças entre utilizadores e itens para recomendação.

Tanto os utilizadores quanto os serviços prestados têm se beneficiado desses tipos de sistemas. A qualidade e o processo de tomada de decisão também melhoraram por meio desses tipos de analises da informação gerada pelos dados.

2.3.1.1. Por que o sistema de recomendação?

- Os utilizadores são ajudados a localizar itens de interesse próprio;
- Por outros lado os fornecedores têm seus itens com o destino correto;
- Maior relevância nos produtos oferecidos;
- Conteúdo personalizado;
- Melhor engajamento;

2.3.1.2. Tipos de sistema de recomendação

- Popular;

É um tipo de sistema de recomendação que funciona com base no princípio da popularidade, em qualquer coisa que esteja na moda. Esses sistemas verificam os produtos ou filmes que estão em alta ou são mais populares entre os utilizadores e os recomendam diretamente. Isso acontece muito de forma informal, quando um conhecido recomenda um livro ou um filme, sem qualquer tipo de tecnologia computacional, apenas utilizando o momento.

- Classificador;

Desta vez utilizando um modelo de *machine learning*, podemos assim prever se o utilizador vai ou não gostar do produto em questão, como mencionamos neste capítulo 2 item 2.2, este modelo utiliza modelos de *machine learning* para realizar as recomendações.

Limitações: dificilmente teremos informações suficientes de todos os utilizadores e ainda com problemas de flexibilidade.

- Conteúdo semelhante;

É o sistema de recomendação que trabalha com o princípio de conteúdo semelhante. Se um utilizador estiver assistindo a um filme, o sistema verificará outros filmes de conteúdo semelhante ou do mesmo gênero do filme que o utilizador está assistindo. Existem vários atributos básicos que são usados para calcular a similaridade durante a verificação de conteúdo semelhante.

Dentre outros, como os já mencionados nos itens 2.2.1 e 2.2.2, filtragem colaborativa e recorrente rede neural, respetivamente.

2.3.2. Alguns problemas que podemos enfrentar

Para a recomendações serem cada fez mais precisa, os utilizadores têm que interagir mais e cada vez mais com os artigos dos *websites*, entretanto, quando um novo artigo for publicado, provavelmente será pouco recomendado. Também temos a preocupação de novos utilizadores não receberem boas recomendações.

3. METODOLOGIA

3.1. DEFINIÇÃO DE PROBLEMA DE NEGÓCIO

Precisamos criar uma estrutura de um *data lake* (Singh, 2019) para que possamos coletar dados em tempo "quase" real, que vem do termo (do inglês, *near real-time*), armazenar e processar, para fazermos as recomendações em *real-time* para os utilizadores dos *websites* (Bhole, Chandrakar e Swarnkar, 2017), lembrando sempre que o que temos é algo muito próximo do *real-time*.

Esses dados já são conhecidos, muito comum e algo muito parecido com arquivos de *logs* de servidores *web*, podemos citar alguns exemplos, tais como *IP* (do inglês, *Internet Protocol*), *timestamp*, *url* (do inglês, *Uniform Resource Locator*) de acesso, isso irá proporcionar a identificação de tráfego entre artigos, é o que esperamos. Temos uma outra opção, para alimentar os algoritmos de *machine learning,* que são os dados em um formato mais agrupado, já processados, tais como, *timestamp*, *url* atual, *url* referência.

3.2. ORGANIZAÇÃO DA ESTRUTURA

Os dados serão gerados a cada acesso em um artigo, iremos então conectar o Kafka a cada *website* da Triangulum para receber esses dados, que por sua vez, ficarão no cluster Kafka e separados por tópicos, por no máximo 24 horas, prontos para serem consumidos.

Ocorrerá dois tipos de consumo dos dados armazenados no *cluster* Kafka, um será para processamento em *real-time* e outro para apenas armazenamento, como veremos na Figura 3.

- Processamento, com o *framework* Apache Spark Streaming (*Spark Streaming | Apache Spark*, 2018), reproduzindo recomendações para os utilizadores dos *websites*, essas por sua vez serão feitas baseado no artigo que está a ser lido, levando em consideração os comportamentos anteriores.

- Tendo em conta que vamos analisar dados em tempo real e sabendo que vamos utilizar o Spark Streaming (Kroß e Krcmar, 2016) para o processamento dos dados, teremos que ter um *data lake* que trabalha de forma distribuída (O'Malley, 2008), isso porque o Spark processa de forma distribuída (Zaharia *et al.*, 2012), tendo assim uma melhor atuação no sistema de recomendação. Nesse ponto criaremos um *cluster* Hadoop (Mehta e Mehta, 2016) para apenas usarmos o HDFS para o armazenamento dos dados.

O processamento de forma distribuída será a forma mais eficaz de processar grandes quantidades de dados.

O Spark Streaming irá iniciar uma sessão para consumir os dados em *real-time* que estão no Kafka, será feito um pequeno processamento nos dados para usarmos o utilizador, também o código lerá o modelo já treinado para por fim usarmos a função "*predict*", vamos chamar assim, como parâmetro de entrada iremos passar o utilizador e quantas recomendações gostaríamos de proporcionar. Logo após essa etapa, vamos gravar na base de dados as recomendações onde o *site* irá fazer uma solicitação das recomendações para preencher os espaços destinados a isso.

O outro consumidor por sua vez e em simultâneo vai armazenar todos os acessos ao *website*, gerando assim uma grande quantidade de dados para que o Spark (Holden Karau e Zaharia, 2017) possa processar e treinar o modelo em um intervalo de tempo pré determinado, por exemplo, a cada hora, iremos definir esse intervalo melhor com a observação da execução do processo.

Abaixo, Figura 3, podemos ver o fluxograma do processo e posteriormente uma definição das tecnologias e seus benefícios, vantagens e desvantagens.

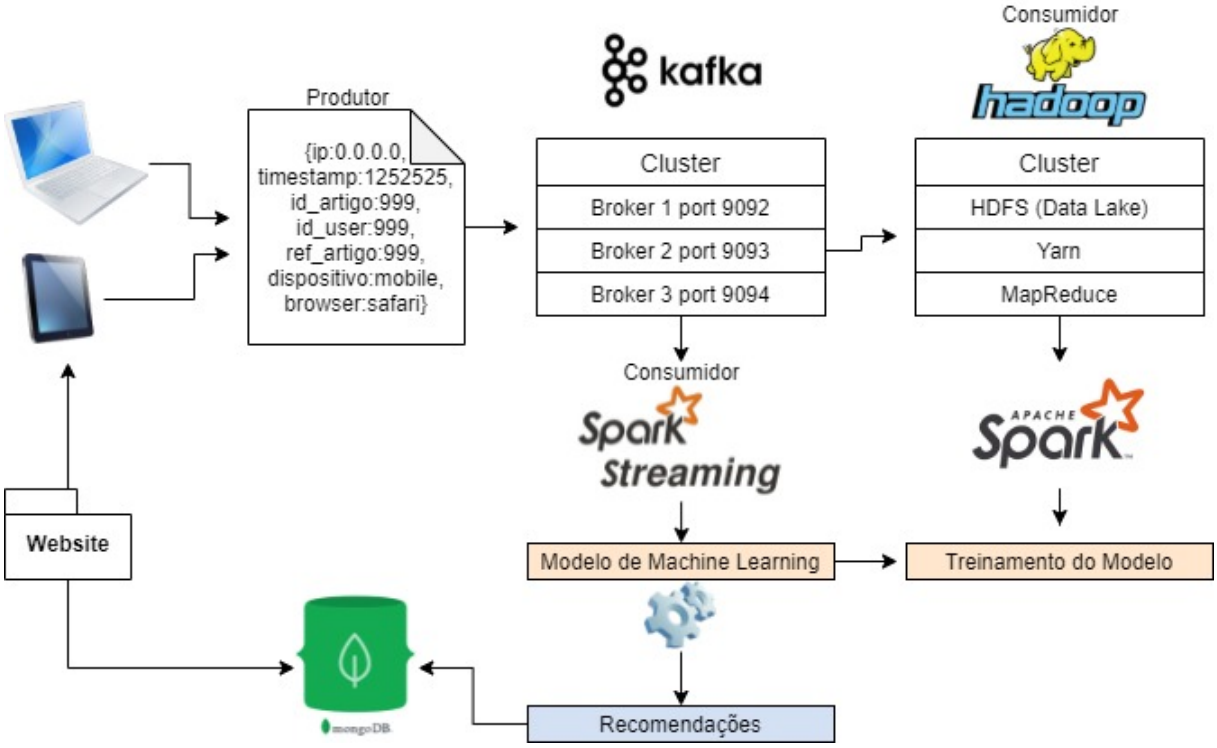

Figura 3 - Fluxograma de toda a estrutura do projeto.

3.3. DATA LAKE STACK

Um *data lake* tem com definição oferecer um armazenamento de dados muito mais econômico do que *data warehouse* (Inmon, 1995), por exemplo. Todavia, com o modelo *schema-on-write* (*Schema-on-Read vs Schema-on-Write - Blog | luminousmen*, 2020) tradicional dos *data warehouses* (Inmon, 1995), o armazenamento da dados é altamente ineficiente mesmo na nuvem.

Grandes quantidades de dados podem ser desperdiçadas devido ao problema de tabela esparsa. Para entender esse problema, vamos imaginar a construção de uma planilha que combina duas fontes de dados diferentes, uma com 20 colunas e outra com 80 colunas, deixando claro que as colunas das tabelas são diferentes. Para concatená-las, é preciso adicionar 80 novas colunas na primeira planilha, aquela com 20 colunas. As linhas da planilha original não possuiriam dados para essas 80 novas colunas e linhas da segunda planilha não teriam dados das 20 colunas originais. O resultado esperado, uma sobrecarga extra de processamento e espaço em disco desperdiçado. Um *data lake* minimiza esse tipo de desperdício. Cada parte dos dados é atribuída a uma única célula isso nos permite que não precisamos combinar os dados na entrada para o *data lake*, não existem linhas

ou colunas vazias. Isso possibilita armazenamento de grandes volumes de dados em menos espaço do que o necessário mesmo para base de dados convencionais relativamente pequenas. Além de precisar de menos armazenamento, quanto o armazenamento e a computação são separados, os clientes podem pagar pelo armazenamento a uma menor taxa na nuvem, independentemente das necessidades de computação. Os provedores de serviços de nuvem, como AWS (https://aws.amazon.com), oferecem até mesmo uma variedade de opções de armazenamento em diferentes faixas de preço.

Outra consideração importantíssima sobre a implementação de um *data lake* em relação a implementação de um *data wearhouse*, foi que nos próximos 10 anos a 12 anos os *data lake* serão prioridades nas companhias e assim como hoje, os dados são como o petróleo, muito mais muito valioso mesmo e cabe aos gestores terem a visão voltada para o futuro, esse que está bem próximo.

3.4. CLOUD OU ON-PREMISES

Considerando a tendência no crescimento do volume de dados, nossa implementação do *data lake* (Giebler *et al.*, 2021; Singh, 2019) para armazenar e processar os dados, será na nuvem AWS. Umas das principais características que nos levou a escolher a nuvem foi a escalabilidade.

Jamais podemos deixar de lado o crescimento do volume dos dados e a escalabilidade nos proporciona adicionar máquinas ao *cluster* com muito mais facilidade do que ter que implementar uma máquina física, onde a implementação pode demorar dias.

Sabemos que a segurança é também muito importante, no nosso ponto de vista um ambiente em nuvem tem uma maior segurança, falando em relação a implementação de um servidor fisicamente e também as replicações de *backup* espalhada em várias regiões, nos deixam mais confortáveis, por sinal são bem feitos, considerando todas as características acima, essas são necessárias para a evolução desses grandes provedores.

Falando em custo fizemos um estudo e verificamos que o *data lake* em nuvem é mais vantajoso, não tendo custos por exemplo de energia elétrica, espaço físico, depreciação dos equipamentos, segurança física dependendo da região que está sua companhia, segurança contra-ataques cibernéticos, dentre outros. Vejamos aqui rapidamente o que precisaríamos para construção do *data lake on premises*, primeiro seria necessário a aquisição de máquinas, no nosso caso seriam três, depois teríamos que ter um espaço físico com refrigeração e rede elétrica para instalação das máquinas, segurança.

Algumas pesquisas e encontramos companhias espalhadas pelo mundo que se especializaram em implementação de *data lakes* em nuvem utilizando um grande provedor de espaço, mas por outro lado, essas usam tecnologias *open source* para a implementação dos mesmos, são elas; Zaloni (https://www.zaloni.com/), Knowledgent (https://www.informatica.com/), o negócio principal dessas companhias é vender o serviço. O cliente por sua vez, na maioria dos casos não está preocupado com as tecnologias utilizadas e sim com o resultado.

3.5. O QUE É UM CLUSTER

Antes de falarmos sobre um *cluster* Hadoop (Borthakur, 2007; Jena *et al.*, 2017), vamos compreender o que é um *cluster* (Nacional, 2014; Li e Peng, 2000). Quando o assunto é computação de alto desempenho, não é difícil pensarmos em servidores, sofisticados e caros respondendo por este trabalho. No entanto, é possível obter resultados tão bons quanto ou superiores a partir de alguma solução de *cluster*, uma tecnologia capaz de fazer computadores mais simples trabalharem em conjunto, como se formassem uma única máquina. Mas o que é um *cluster*? *Cluster* (ou clustering) (Paas *et al.*, 2016) é, em poucas palavras, o nome dado a um sistema que relaciona dois ou mais computadores para que estes trabalhem de maneira conjunta no intuito de processar uma tarefa (Nacional, 2014). Estas máquinas dividem entre si as atividades de processamento e executam este trabalho de maneira simultânea. Cada computador que faz parte do *cluster* recebe o nome de nó (ou *node*). Teoricamente, não há limite máximo de nós, mas independentemente da quantidade de máquinas que o compõe, o *cluster* deve ser "transparente", ou seja, ser visto pelo utilizador ou por outro sistema que necessita deste processamento como um único computador. Os nós do *cluster* devem ser interconectados, preferencialmente, por uma tecnologia de rede conhecida, para fins de manutenção e controle de custos, como a Ethernet. É extremamente importante que o padrão adotado permita a inclusão ou a retirada de *nodes* com o *cluster* em funcionamento, do contrário, o trabalho de remoção e substituição de um computador que apresenta problemas, por exemplo, faria a aplicação como um todo parar. A computação em *cluster* se mostra muitas vezes como uma solução viável porque os *nodes* podem até mesmo ser compostos por computadores simples, como PCs (do inglês, *Personal Computer*) de desempenho mediano. Juntos, eles configuram um sistema de processamento com capacidade suficiente para dar conta de determinadas aplicações que, se fossem atendidas por supercomputadores ou servidores sofisticados, exigiriam investimentos muito maiores. Não é necessário haver um conjunto de *hardware* exatamente igual em cada nó. Por outro lado, é importante que todas as máquinas utilizem o mesmo sistema operacional, de forma a garantir que o *software* que controla o *cluster* consiga gerenciar todos os computadores que o integram. As tecnologias de *clustering* possibilitam a solução de diversos problemas que envolvem grande volume de processamento. As aplicações que um *cluster* pode ter são diversas, indo desde a simples melhora no desempenho de um determinado sistema ou a hospedagem de um *site*, até o processo de pesquisas científicas complexas. O que realmente chama a atenção, é que todo o processamento pode ser feito de maneira que pareça ser um único computador dotado de alta capacidade. Assim, é possível que determinadas aplicações sejam implementadas em *cluster*, mas sem interferir no funcionamento de outras aplicações que estejam relacionadas. A origem da denominação "*cluster*" não é clara, mas sabe-se que as primeiras soluções de processamento paralelo remontam à década de 1960, havendo, a partir daí, alguns princípios que hoje formam a base da ideia de *clustering*. O fato é que o passar do tempo não torna o conceito ultrapassado. Há um motivo especial para isso: os *clusters* se relacionam intimamente à otimização de recursos, uma necessidade constante em praticamente qualquer cenário computacional. E este aspeto pode se tornar ainda mais atraente quando a ideia de *cluster* é associada a conceitos mais recentes, como *cloud computing* e Virtualização.

3.6. ESCOLHA DO DATA STORAGE

O HDFS (Jena *et al.*, 2017) permite gerenciar várias máquinas com se fosse uma, faz parte do ecossistema Hadoop e é a principal ferramenta para construção do nosso *data lake*, onde tudo é analisado de forma distribuída com o custo zero, sendo que as ferramentas são *open source* (*What is open source software? | Opensource.com*, 2021).

Não só pelo custo muito pequeno, onde temos que levar em conta o custo de aprendizagem e desenvolvimento, vamos usar HDFS (Borthakur, 2007), isso permite que o processamento em *cluster* transforme os dados em um formato desejável em uma questão de tempo mínima. Útil para utilizadores corporativos e cientistas de dados, sendo assim garantido que os dados sejam transformados e processados de acordo com os objetivos de análise. É necessário que, durante o processo de inserção de dados no *data lake*, o não tome decisões precipitadas sobre como transformar ou padronizar os mesmos. Em vez disso, tomamos esta decisão apenas quando nós formos fazer a leitura dos dados. Nesse ponto, os dados do *data lake* (Giebler *et al.*, 2021) possuem uma variedade de formas e vamos utilizar diversas ferramentas para padronizar ou transformar os dados. Um dos maiores benefícios dessa metodologia é que diferentes utilizadores de negócios podem realizar diferentes padronizações e transformações, dependendo de suas necessidades exclusivas. Ao contrário de um *data warehouse* tradicional, os utilizadores não estão limitados a apenas um conjunto de padronizações e transformações de dados que devem ser aplicadas na abordagem convencional de esquema por gravação. Nesse momento, também podemos provisionar fluxos de trabalho para processamento a repetitivos de dados. Ferramentas apropriadas podem processar dados para casos de uso em lote e quase em tempo real. O processamento em lote é para cargas de trabalho tradicionais de extração, transformação e carregamento (ETL) (do inglês, *Extract Transform Load*) (https://pt.wikipedia.org/wiki/Extract,_transform,_load), por exemplo, nós podemos querer processar informações diversas para gerar um relatório operacional diário.

Streaming é para cenários em que o relatório precisa ser entregue em tempo real ou quase em tempo real e não pode esperar por uma atualização diária. Por exemplo, uma grande companhia de *courier* pode necessitar de dados de *streaming* para identificar os locais atuais de todos os seus caminhões em um determinado momento. Diferentes ferramentas são necessárias, dependendo se o seu caso de uso envolve *batch* ou *streaming*. Para casos de uso em *batch*, as organizações geralmente usam Pig (https://pig.apache.org/), Hive (https://hive.apache.org/), Spark (https://spark.apache.org/) (Drakonaki e Allen, 2010) e MapReduce (https://hadoop.apache.org/). Para casos de uso de *streaming*, podemos provavelmente usar ferramentas diferentes, como Spark Streaming, Kafka (https://kafka.apache.org/) Flume (https://flume.apache.org/) e Storm (https://storm.apache.org/). Podemos ainda usar outras ferramentas de integração como Apache Drill (https://drill.apache.org/), Apche NiFi (https://nifi.apache.org/), Apache Beam (https://beam.apache.org/) e Apache Sqoop (https://sqoop.apache.org/), na Tabela 1, abaixo segue uma breve descrição dessas ferramentas da Apache.

Ferramenta Open Source	Descrição
Apache Beam	Serviço de processamento de dados
Apache Drill	Linguagem de programação para o Hadoop
Apache Flume	Serviço de *etl* em *streaming* de dados
Apache Hive	Manipula os dados no HDFS com um *data warehouse*
Apache Kafka	Serviço de mensageria, *elt* em *streaming* de dados
Apache MapReduce	Processamento de *bid data*, trabalha de forma distribuída em disco
Apache NiFi	Serviço de *etl* em dados *batch*
Apache Pig	Linguagem de programação, baseada no SQL
Apache Spark	Processamento de *bid data*, trabalha de forma distribuída em memória
Apache Spark Streaming	Serviço de processamento de *streaming* de dados
Apache Sqoop	Serviço de *etl* em dados *batch* via jdbc
Apache Storm	Serviço de *etl* com processamento de dados

Tabela 1 – Ferramentas *open source* da Apache

Vejamos como podemos criar nosso estrutura do Hadoop HDFS (Borthakur, 2008) composto por n números de nós, temos o seguinte esquema, ilustrado na Figura 4, abaixo.

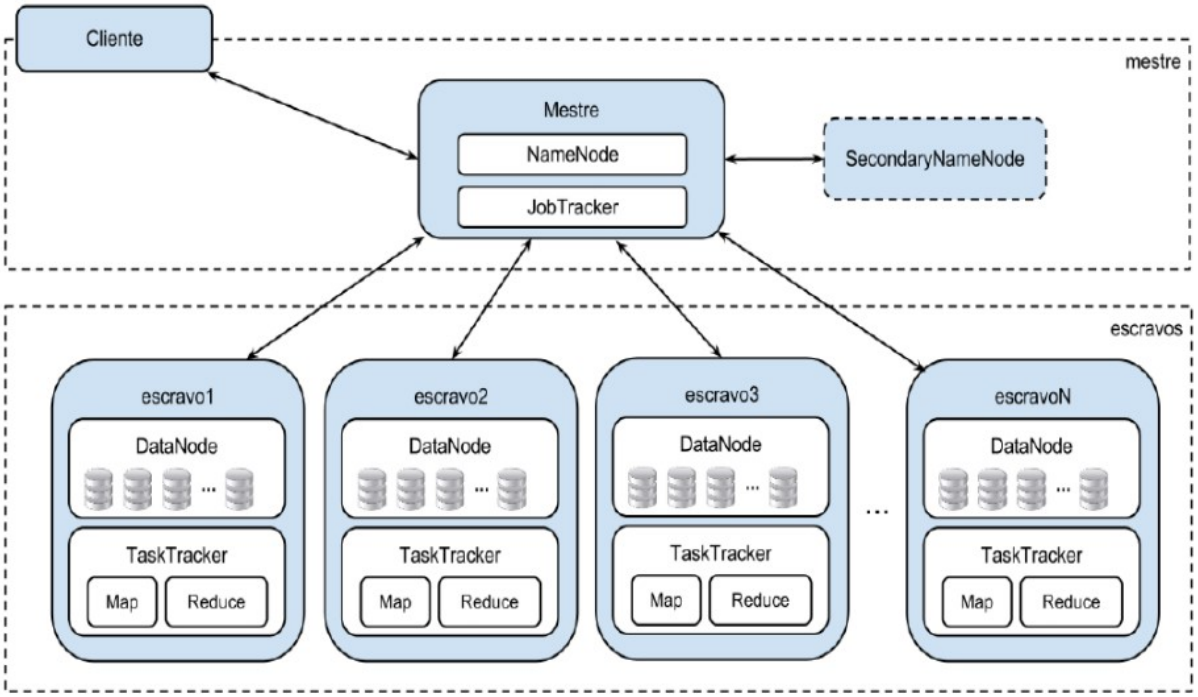

Figura 4 - Organograma de uma estrutura de *clusters* de n máquinas. (Da, Ti e Negócios, 2011)

O HDFS um sistema de gerenciamento de arquivo com uma característica diferenciada, de forma distribuída. Ele possui as mesmas características de um sistema de arquivo normal, entretanto ele deve permitir o compartilhamento em diversos *hardwares* diferentes, com manipulação remota e promove escalabilidade, para quando for necessário incluir outras máquinas no *cluster*. Como podemos ver na Figura 4, o HDFS é composto por um NameNode, que o Mestre do *cluster* e pelos DataNode, que são os trabalhadores, por sua vez o NameNode é a máquina mais robusta do *cluster* para poder gerenciar os trabalhadores e fazer a distribuição dos arquivos pelos mesmo. Principais características:

- Controlar o acesso ao disco rígido,
- Tolerante a falhas, com a falha em um dos *cluters* ela não interrompe o sistema.
- Integridade, ele controla as operações realizados nos arquivos, como escrita e remoção, sempre com permissão.
- Segurança, com gerenciamento de acesso e controlo de privacidade.
- Desempenho alta por ter baixos números de utilizadores, ou seja, para uso interno.
- Consistência, todos utilizadores devem ter a mesma visão dos arquivos.

3.6.1. Object storage

Temos outras opções de *data storage* que podemos usar, mas por ter um custo a mais deixamos essas opções em *stand by*, tais opções são: AWS S3 (do inglês, *Simple Storage Service*) (https://aws.amazon.com/pt/s3/), Microsoft Azure (https://azure.microsoft.com/pt-pt/), Google Cloud (https://cloud.google.com/).

3.7. FUNCIONAMENTO DO HADOOP

Requisição do cliente é o primeiro passo, isso gera um *job* e assim começa o processamento do Hadoop (Fay, 2011), se o *job* foi para execução de MapReduce, o NameNode aciona os DataNode para ver a alocação dos arquivos. Se for apenas um trabalho de escrita de dados o NameNode trabalha junto com os DataNode. No nosso projeto o modo de execução será o totalmente distribuído para o processamento da aplicação, Hadoop estará em um *cluster* de computadores real. Nessa opção, como no modo pseudo-distribuído, também é necessário editar os três arquivos de configuração, definindo parâmetros específicos e a localização do SecondaryNameNode e dos nós *workers*. entretanto, como nesse modo temos diversos computadores, devemos indicar quais máquinas irão efetivamente executar cada componente.

3.8. YARN

O Apache YARN (do inglês, *Yet Another Resource Negotiator*) (Kulkami e Khandewal, 2014) (https://hadoop.apache.org/) é a camada de gerenciamento de recursos do Hadoop. O YARN foi introduzido no Hadoop 2.x. e agora melhorado no Hadoop 3.x. O YARN permite diferentes mecanismos de processamento de dados, como processamento de gráficos, processamento interativo, processamento de fluxo e processamento em lote para executar e processar dados armazenados no HDFS. Além do gerenciamento de recursos, o YARN também é usado para agendamento de trabalhos. O YARN amplia o poder do Hadoop para outras tecnologias em evolução, para que eles possam aproveitar as vantagens do HDFS e do *cluster* econômico. O Apache YARN também é considerado como o sistema operacional de dados do Hadoop 2.x. A arquitetura baseada em YARN do Hadoop fornece uma plataforma de processamento de dados de uso geral que não se limita apenas ao MapReduce. Ele permite que o Hadoop processe outro sistema de processamento de dados para fins específicos, diferente do MapReduce. Ele permite executar várias estruturas diferentes no mesmo *hardware* em que o Hadoop é implementado.

Como surgiu o YARN? Junto com o HDFS, o MapReduce tem sido a base de computação em larga escala na última década, para aplicações analíticas de *big data*. Com o tempo o Hadoop amadureceu se tornando um ambiente computacional estável e genérico o suficiente para o processamento de praticamente qualquer aplicação. No entanto, à medida que o Hadoop passou a ser adotado de forma mais ampla, outras especializações se tornaram necessárias e ficou evidente que o MapReduce não era muito adequado para processamento iterativo, típico de aplicações de *machine learning*. O MapReduce é paralelizável, fácil de entender, e compreende basicamente 2 processos: mapeamento e redução e oculta os detalhes da computação distribuída, permitindo assim a construção de aplicações relativamente fáceis para processamento em larga escala. No entanto, para conseguir coordenação e tolerância a falhas, o MapReduce utiliza um modelo de execução de extração de dados que exige escritas intermediárias de volta no HDFS. Essa escrita frequente em disco, representa custo de tempo em qualquer sistema de computação; como resultado, embora seja extremamente seguro e resiliente, o MapReduce também é mais lento em cada tarefa de Mapeamento e Redução. Pior ainda, quase todas as aplicações precisam encadear vários *jobs* de MapReduce em muitos passos, criando um fluxo de dados em direção ao resultado final desejado. Isso resulta em quantidades enormes de dados intermediários escritos no HDFS, que não são necessários ao utilizador, gerando custos adicionais no que diz respeito ao uso de disco. Para tratar esses problemas, o Hadoop passou a usar um *framework* mais genérico de gerenciamento de recursos e processamento, o YARN. Anteriormente a aplicação MapReduce alocava os recursos (processadores, memória) aos *jobs* especificamente. O YARN oferece um acesso mais genérico aos recursos para aplicações Hadoop. O resultado é que ferramentas especializadas não precisam ser decompostas em uma série de *jobs* de MapReduce e podem ser mais complexas. Ao generalizar o gerenciamento do *cluster*, o modelo de programação inicialmente pensado para o MapReduce pôde ser expandido para incluir novas abstrações e operações. O YARN é uma evolução do MapReduce onde as funções do JobTracker são repartidas em *daemons* independentes. Uma das funções principais do MapReduce é a de partilhar os dados para as funções de Map e Reduce, a outra função é gerenciar as falhas e procurar nós disponíveis para executar a função onde houve falha. Para isso o YARN muda um pouco a nomenclatura do nó *master* e o apelida de RM (do inglês, *Resource Manager*) ou AM (do inglês, *Application Master*), onde cada função MapReduce é uma aplicação definida pelo nó mestre e *o resource manager* fica responsável por reordenar os nós no caso de

falhas dos nós escravos. Com isso, o YARN agora faz a gestão do MapReduce, através de 2 processos: o *resource manager* e o *application master*. Na versão 3 do Hadoop, o YARN ganhou melhorias e correções, consolidando assim sua função de gestão de recursos do *cluster* Hadoop.

3.9. CLUSTER HADOOP

3.9.1. Recomendações de hardware para um cluster Hadoop

Segundo o curso *online* de DSA (https://www.datascienceacademy.com.br/), módulo de *data lakes,* as configurações abaixo, são adequadas para o começo de um grande projeto.

3.9.1.1. Slaves ou Workers

- 12 discos SATA 7200 RPM de 1 TB cada;
- 48 GB de memória RAM cada;
- Processador com *clock* médio e menos de 2 *sockets*
- Rede requer *links* de 1 GB para todos os nodes

3.9.1.2. Master

- Pelos menos 4 volumes de *storage* redundantes
- 64 GB de RAM
- 24 CPU's

Estas são apenas recomendações e que podem variar de acordo com os fatores para o planejamento do *cluster*: objetivo, serviços e *layout*. Para o nosso *data lake* vamos utilizar três máquinas, mas não com essas configurações. A descrição das máquinas será listada a seguir para o NameNode e para os DataNode

3.9.2. Hardware para nosso cluster Hadoop

Nossas máquinas serão:

NameNode; Amazon Linux 2, que são versão do CentOS, será m4.large, com duas CPU's e 8GB de mémoria RAM, *network* VPC padrão, *subnet* us-east-2a, com muita atenção em relação a *subnet*, as três máquinas tem que estarem na mesma *subnet* e ao criar a instância essa configuração não pode ser alterada, *shutdown* será por padrão *stop*, assim que quando encerrar a máquina ela só irá parar, *tenancy* será *shared* que significa uma única máquina física com várias máquinas EC2 do mundo inteiro rodando, teremos 8GB em espaço em disco, a princípio será suficiente, mas com a flexibilidade que temos poderemos adicionar mais espaço ao decorrer do projeto, vamos inserir *tags*

com a chave DataLake e o valor NameNode, vamos criar um grupo de segurança do zero, esse grupo cria automaticamente uma regra de segurança para o acesso via SSH (do inglês, *Secure Shell*), nós vamos usar um terminal para facilitar as configurações e para isso vamos adicionar uma regra *all trafic*, vamos usar essa regra somente o período de configuração das máquinas, apesar somente poder ser acessada com senha, custo dessa máquina é de $0,10 por hora. Tudo pronto para iniciar a máquina, lembrando que será solicitado para criar uma chave de segurança, detalhe, sem essa chave não é possível acessar as instâncias via SSH, então criamos a chave de segurança e guardamos em um lugar apropriado.

Os DataNodes serão t2.medium, com duas CPU's e 4GB de mémoria RAM, *network* VPC padrão, *subnet* us-east-2, demas configurações igual a o NameNode, custo dessa máquina é de $0,0464 por hora e vamos usar a mesma chave de segurança e o mesmo grupo de segurança.

Depois da inicialização vamos renomear as máquinas como: o NameNode será hadoopmaster e os DataNode serão hadoopslaves1 e hadoopslaves2

Ao construir seu ambiente na nuvem AWS, certifique-se que configurar os grupos de segurança para que o acesso seja permitido apenas a partir do seu endereço IP. Isso evita que seu ambiente seja invadido, comprometendo assim o custo do seu ambiente em nuvem.

Como as instâncias Linux EC2 não tem interface gráfica vamos fazer conexão via SSH, para o nosso caso estamos usando Windows, nessas condições temos que fazer a conexão com Putty ou Moba, vamos conectar com o Moba via SSH para fazermos as configurações.

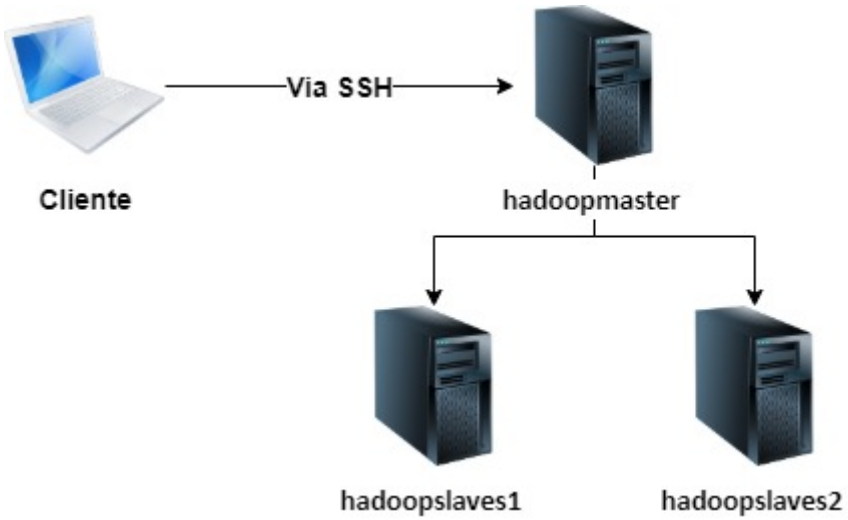

Figura 5 – Cluster hadoop conexão via ssh com o cliente.

3.10. APACHE KAFKA

Analisando as ferramentas disponíveis no mercado, considerando novamente o recurso financeiro um problema, vamos optar por um serviço *open source*, a ferramenta mais interessante dentre muitas opções será o Kafka (Romero e Oliveira, 2011). O Kafka é uma feramenta de distribuição de

dados de forma distribuída e escalável, que nos permite conectar em uma ou mais fontes de dados, guardar essas mensagens/dados e disponibilizar para o consumo.

O Kafka funciona como *meedlewaer*, ou seja, responsável pela camada de mensagens da nossa estrutura de processamento de dados em tempo real, onde a produção de mensagens ocorrerá de forma contínua por uma ou mais fontes de dados, essas mensagens serão armazenadas no *cluster* Kafka. Um *cluster* Kafka é gerenciado pelo Zookeeper, podemos ter vários *brokers*, onde cada *broker* é um nó do cluster para que os dados sejam armazenados de forma distribuída, para que os dados sejam guardados para depois possam ser consumidos, precisamos criar os tópicos, cada tópico criado pode ser específico para cada tipo de dado, no momento da criação dos tópicos, podemos determinar o fator de replicação, tornando assim o *cluster* tolerante a falhas.

Os dados ficarão no *cluster*, mesmo após ser consumido, por um período configurável, sendo assim mais uma vez tolerante a falhas, no caso específico aqui implementado, o consumo ocorrerá de forma em *streaming*, mas podendo ser consumido em outro período de acordo com a necessidade.

Vamos citar os principais motivos que levaram a escolha dessa ferramenta:

- Alto rendimento: o Kafka é capaz de lidar com dados de alta velocidade e alto volume usando *hardware* modesto. É capaz de suportar o processamento de milhares de mensagens por segundo.

- Baixa latência: o Kafka é capaz de lidar com mensagens de latência muito baixa em intervalo de milissegundos, exigida pela maioria dos casos de uso para aplicações *real-time*.

- Tolerante a falhas: a capacidade inerente do Kafka de ser resistente a falhas de nó/máquina dentro de um *cluster*.

- Durabilidade: os dados/mensagens são persistentes no disco, tornando-os duráveis e as mensagens também são replicadas para que nunca sejam perdidas.

- Escalabilidade: o Kafka pode ser expandido *on-the-fly*, sem incorrer em tempo de inatividade, adicionando novos nós. A manipulação de mensagens dentro do *cluster* Kafka é totalmente transparente e elas são perfeitas.

- Distribuído: suporta inerentemente arquitetura distribuída, tornando-o escalável usando recursos como replicação e particionamento. Recursos do *message broker*.

- Alta simultaneidade: capaz de lidar com milhares de mensagens por segundo e também em condições de baixa latência com alta taxa de transferência. Kafka permite a leitura e escrita de mensagens em alta simultaneidade.

Por padrão persistente: por padrão, as mensagens são persistentes, tornando-as duráveis e confiáveis.

- Consumidor: Kafka pode ser integrado com uma variedade de consumidores. Cada cliente tem uma capacidade diferente de lidar com essas mensagens provenientes do Kafka e, por causa de sua capacidade inerente de persistência, pode se comportar ou agir de maneira

diferente, de acordo com o consumidor com o qual se integra. Ele também se integra bem com uma variedade de consumidores escritos em várias linguagens de programação.

- Capacidade de manipulação de lotes (funcionalidade semelhante a ETL): como o Kafka persiste mensagens, ele também pode ser empregado para casos de uso semelhantes processamento em lotes e também pode fazer o trabalho de um ETL tradicional.

- Capaz de lidar com uma variedade de casos de uso comumente necessários para um *data lake*, ou seja, agregação de registros, rastreamento de atividades da *web* e assim por diante.

- Projetado para funcionar em *hardware* de *commodity*. Para as POCs (do inglês, *Proof of Concept*), tudo bem usar um *hardware* simples, mas para um ambiente de produção, não é recomendável selecionar um *hardware* comum, embora ele funcione muito bem.

Ajuda a gerenciar o *pipeline* de dados em tempo real. Este é um dos principais motivos de nossa escolha, pois precisamos encontrar uma peça de tecnologia para lidar com mensagens em tempo real.

- Apache Kafka (Romero e Oliveira, 2011) é *open source* e tem uma grande comunidade de seguidores. Além disso, existem muitas companhias que estão prontas para fornecer suporte comercial, o que pode ser um aspecto importante para as grandes companhias devido à sua importância.

- Kafka funciona bem com o Apache Spark (Zaharia *et al.*, 2012), nossa solução para processamento de dados em tempo real.

Para transmitir mensagens aos consumidores, ele usa recursos do sistema operacional, por isso, pode funcionar muito bem. Ele também usa muitos recursos do sistema operacional para fazer muitas coisas com eficiência, e esse é definitivamente um ponto positivo.

3.11. APACHE SPARK

Como vimos no tópico anterior, o Kafka funciona muito bem com o Spark, aproveitando essa compatibilidade importante entre as ferramentas vamos usar o Spark para a camada de processamento dos dados, uma vez que o Spark processa os dados em memória até 100 vezes mais rápido que o Hadoop MapReduce, usufruindo também da memória combinada do *cluster* Hadoop. O Spark possui várias funcionalidades, incluindo o Spark Streaming, que vamos utilizar aqui no projeto, vejamos a seguir em detalhes.

3.11.1. O que é Spark Streaming?

O Apache Spark Streaming (Kroß e Krcmar, 2016) é um sistema de processamento de *streaming* tolerante a falhas escalonável que oferece suporte nativo a cargas de trabalho em *batch* e *streaming*. Spark Streaming é uma extensão do Spark API principal que permite que engenheiros e cientistas de dados processem dados em tempo real de várias fontes, incluindo (mas não se limitando a) Kafka,

Flume e Amazon Kinesis. Esses dados processados podem ser enviados para sistemas de arquivos, bancos de dados e painéis ativos. Sua abstração principal é um fluxo discretizado ou, em resumo, um *dstream*, que representa um fluxo de dados dividido em pequenos lotes. *Dstreams* são construídos em RDDs (do inglês, *Resilient Distributed Datasets*), a abstração de dados principal do Spark. Isso permite que o Spark Streaming se integre perfeitamente a qualquer outro componente do Spark, como MLlib e Spark SQL. O Spark Streaming (Zaki *et al.*, 2020) é diferente de outros sistemas que têm um mecanismo de processamento projetado apenas para *streaming* ou têm APIs de *batch* e *streaming* semelhantes, mas compilam internamente para mecanismos diferentes. O mecanismo de execução único do Spark e o modelo de programação unificado para *batch* e *streaming* levam a alguns benefícios exclusivos em relação a outros sistemas de *streaming* tradicionais.

Quatro aspectos principais do Spark Streaming

- Recuperação rápida de falhas e atrasos
- Melhor balanceamento de carga e uso de recursos
- Combinação de *streaming* de dados com conjuntos de dados estáticos e consultas interativas
- Integração nativa com bibliotecas de processamento avançado (SQL, *machine learning*, processamento de gráfico)

Uma breve visão do que amos fazer, abrir conexão do Spark Streaming, conectar no Kafka para consumir os acessos dos utilizadores nos *websites* e baseado no artigo iremos, em *real-time*, processar esses dados e produzir recomendações para o mesmo, proporcionando assim maior tempo de leitura e mais visualizações nas páginas dos *websites* da Triangulum.

3.12. PASSOS PARA A CONFIGURAÇÃO DOS NODES

Instalação do Java JDK 1.8 nas três máquinas configuração das variáveis de ambiente do java para o utilizador *root*, no nosso caso estamos usando ec2-user, editando o .bash_profile, acresentando três linhas com as configurações do Java.

Criar um utilizador Hadoop e dar privilégios de administrador, assim não precisaremos usar o utilizador ec2-user, façamos isso em todas as máquinas.

Configurar o SSH em todas as máquinas, para que o acesso seja sem senha, vamos fazer algumas alterações no ficheiro sshd.config para habilitar a porta 22 e acesso por chave pública, também precisamos criar com utilizador hadoop a chave de segurança "*keygen*" e copia-la para todos os nós do *cluster*.

Instalar o hadoop-3.1.0 como utilizador hadoop apenas no NameNode, configurar das variáveis de ambiente do Hadoop para o utilizador hadoop também editando o .bash_profile, configurar vários arquivos de configuração do Hadoop. Feito isso vamos fazer uma cópia segura via scp (do inglês, *Secure Copy*) para os DataNodes.

Instalar o spark-2.3.0 como utilizador hadoop e configurar das variáveis de ambiente do Spark para o utilizador hadoop, é uma boa prática sempre editar o .bash_profile, assim como o Kafka.

3.13. FUNCIONAMENTO DO KAFKA

Iremos configurar o Kafka *producer* para receber os dados de *streaming* de cliques nos *websites*, esse *producer* envia os dados para o tópico que será distribuído pelos *brokers*, temos 3 *broker* sendo 1 em cada *node*, como já falamos em capítulos anteriores, o *cluster* Kafka é gerenciado pelo Zookeeper, vamos configurar o *cluster* Kafka para guardar os dados em seu *cluster* por apenas 24 horas, por padrão o *cluster* Kafka retém os dados por 168 horas. Seguindo o ciclo dos dados em *real-time*, o Spark irá receber os dados e processar as recomendações, como podemos ver na Figura 6.

Figura 6 – Ciclo, utilizador, Kafka, Spark, recomendações.

Ainda na camada de mensagem, criamos 2 *consumer*, sendo um para guardar os dados no HDFS e o outro para passar os dados de *streaming* ao Spark, os dados no HDFS serão guardados com um propósito definido, dados para treinamento do algoritmo de recomendação, onde vamos treinar a cada 1 hora, como vemos na Figura 7 abaixo:

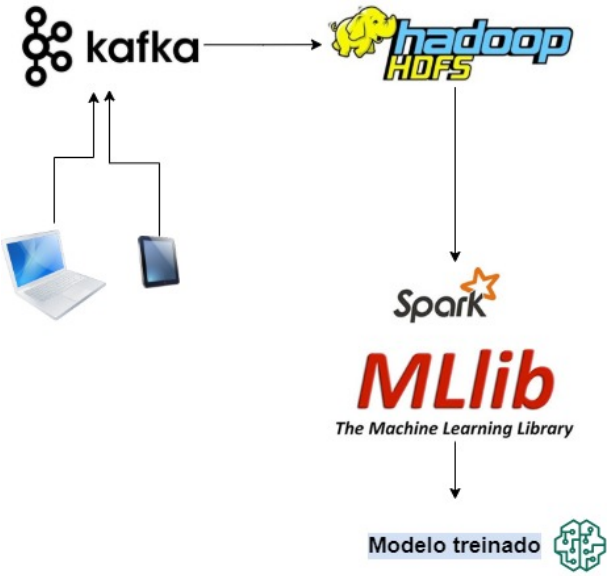

Figura 7 – HDFS, consumidor para armazenamento e MLlib para treinamento do modelo.

O Spark por sua fez vai conectar no HDFS e realizar o processamento dos dados a cada 1 hora, deixando assim o modelo atualizado para fazer as recomendações em tempo real, essas recomendações serão enviadas como resposta para o *site* onde o utilizador está a navegar.

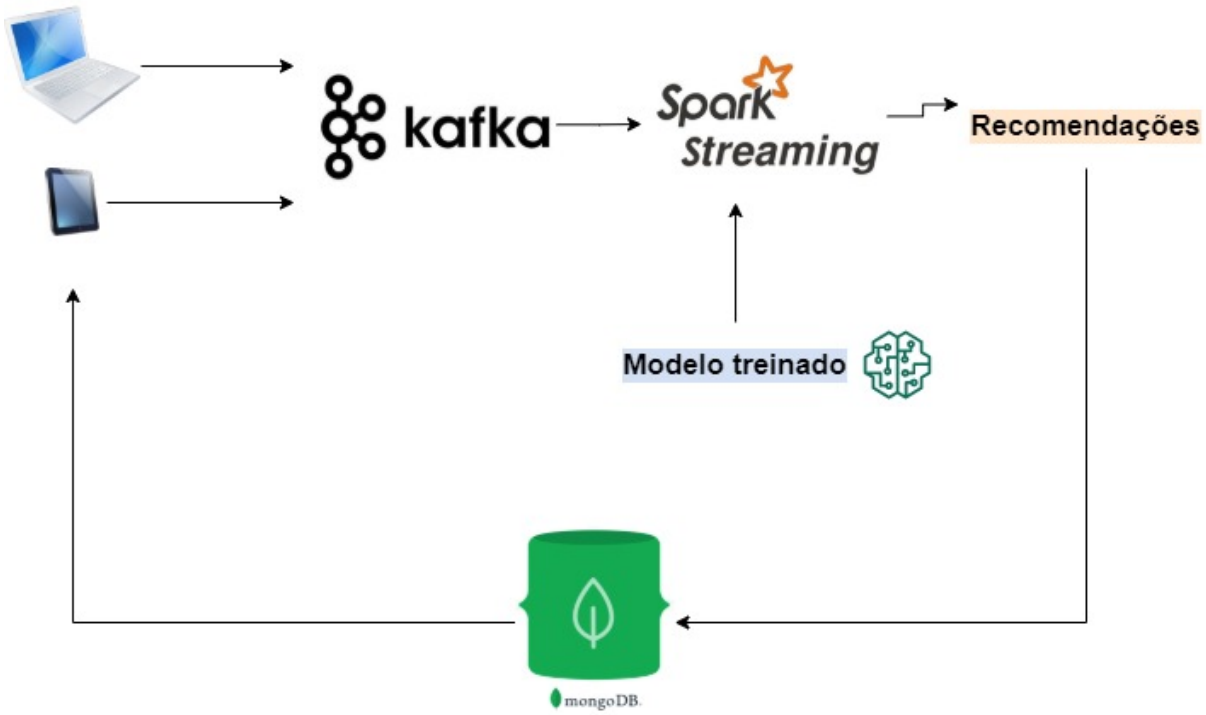

Figura 8 – *Streaming*, consumidor para recomendações *real-time*.

4. RESULTADOS E DISCUSSÃO

4.1. EXEMPLO DE RECOMENDAÇÃO

O utilizador chega ao *website*, por exemplo, um *site* de vendas de produtos, para ficar claro vamos usar como modelo o *site* da amazon.com, que é um dos maiores *sites* de venda de produtos *online* e usa algoritmos de recomendação. Os algoritmos utilizados pela Amazon, estão destacados no capítulo 2, item 2.2.

A entrada no site não nos importa no caso, então o utilizador encontra seu produto e está na seguinte página, Figura 9.

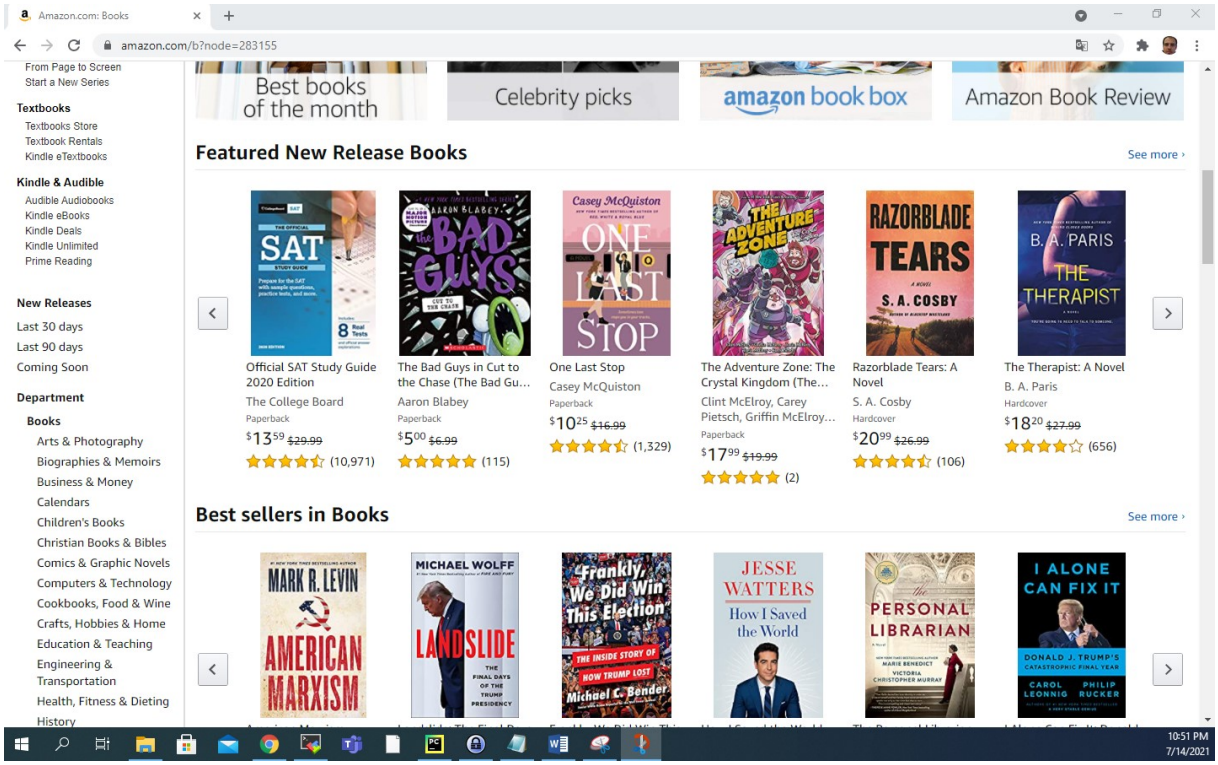

Figura 9 - Website modelo de compra de produtos *online*. (https://www.amazon.com/)

Por sua vez, o utilizador seleciona o livro desejado e então estamos a ver um produto em especifico.

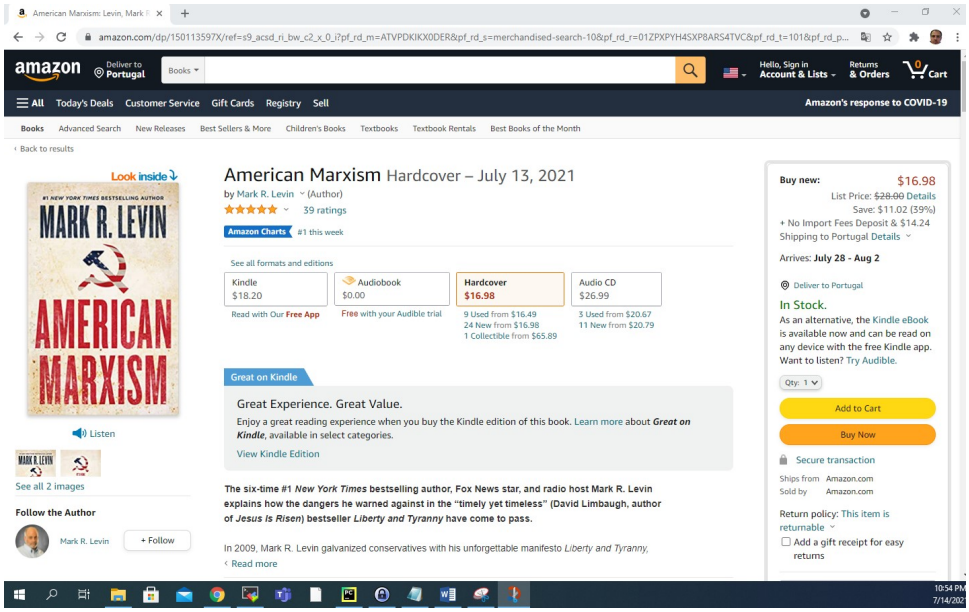

Figura 10 - Produto selecionado. (https://www.amazon.com/)

Logo vemos a descrição do produto e características um pouco mais abaixo na página, encontramos muitas recomendações, como por exemplo, frequentemente são comprados juntos, quem viu esse também viu esse outro, ou quem comprou esse também comprou esse outro, com vemos abaixo Figura 11.

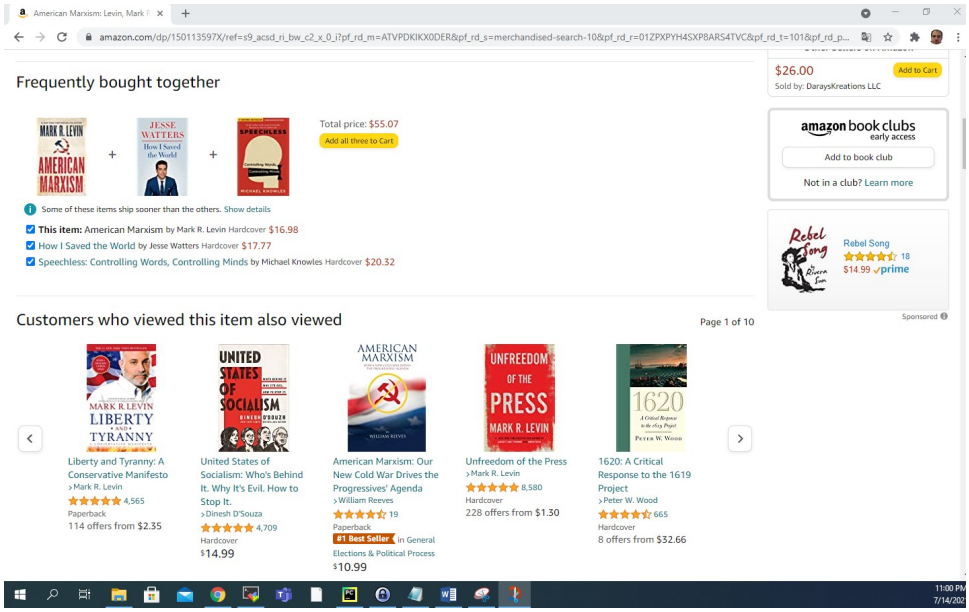

Figura 11 - Recomendações baseadas no produto selecionado. (https://www.amazon.com/)

Essa é a principal ideia do projeto, mas para chegarmos a isso, custou muito, a seguir vejamos tudo que foi necessário para a construção do projeto e como produzimos as recomendações.

4.2. KAFKA PRODUCER

Fizemos uma simulação do dados em *real-time* (Kroß e Krcmar, 2016) com um produtor de acessos em um respectivo *webside*, para isso usamos um código em Java. Este simula um utilizador acessando o *website*, produzindo assim uma informação do respectivo acesso.

Os *streaming* de dados são produzidos e a cada segundo um acesso é simulado e entra no Kafka e é direcionado ao tópico que criamos e assim fica a espera de ser consumido pelos *consumers*, Figura 12.

Figura 12 – Kafka tópicos

Foram utilizadas as bibliotecas disponíveis em Java, que proporcionaram o desenvolvimento de um Kafka Producer, um produdor de mensageria, as bibliotecas foram:

- org.apache.kafka.clients.producer.*;
- java.text.*;
- java.util.*;

```
public class KafkaProduceApp {
    public static void main(String[] args) {

        // Cria a classe Properties para instanciar o Producer com as configurações desejadas:
        Properties props = new Properties();

        props.put("bootstrap.servers", "192.168.1.151:9092, 192.168.1.151:9093, 192.168.1.151:9094");
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
```

Figura 13 - Kafka *producer IPs brokers*.

4.3. KAFKA CONSOLE CONSUMER

Como podemos ver abaixo, Figura 14, o *streaming* de mensagens sendo testada com *shell script* fornecido pelo próprio Kafka, este teste faz uma conexão no tópico onde as mensagens estão chegando, isso permite a visualização em *real-time*, quase 12 acessos por segundo, como codificamos no *script* Java acima mencionado.

As informações consumidas são: Datetime, que representa o momento exato do acesso ao *site* da Triangulum, seguido pelo *user* que são os utilizadores que estão a acessar as *webpages*, o *item*, por sua vez é o artigo que está a ser lido e ainda temos algumas informações adicionais, tais como, de quais navegador e dispositivo o utilizador estava a utilizar para chegar aos *websites* da Triangulum.

```
{datetime: 2021/07/23 09:26:20:415, user: 534, item: 717, device: mobile: browser: chrome}
{datetime: 2021/07/23 09:26:20:501, user: 487, item: 910, device: mobile: browser: chrome}
{datetime: 2021/07/23 09:26:20:588, user: 297, item: 554, device: mobile: browser: chrome}
{datetime: 2021/07/23 09:26:20:674, user: 976, item: 492, device: mobile: browser: chrome}
{datetime: 2021/07/23 09:26:20:761, user: 410, item: 542, device: mobile: browser: chrome}
{datetime: 2021/07/23 09:26:20:847, user: 499, item: 517, device: mobile: browser: chrome}
{datetime: 2021/07/23 09:26:20:934, user: 655, item: 451, device: mobile: browser: chrome}
{datetime: 2021/07/23 09:26:21:020, user: 126, item: 235, device: mobile: browser: chrome}
{datetime: 2021/07/23 09:26:21:106, user: 964, item: 592, device: mobile: browser: chrome}
```

Figura 14 - Kafka console consumer.

4.4. INSERÇÃO DOS DADOS NO DATA LAKE COM O STREAMSETS

Nessa etapa do projeto, estamos a armazenar os dados produzidos pelo Kafka Producer em nosso HDFS, no Hadoop, o Hadoop foi concebido para armazenamento de grande quantidade de dados e de forma distribuída, assim podemos fazer a leitura dos mesmos e processar com o Spark para treinar o algoritmo de *machine learning*.

Usamos aqui componente que facilita nosso armazenamento no HDFS (Borthakur, 2008), o Streamsets.

O Streamsets (https://streamsets.com/), é um *framework open source* para desenvolvimento de *pipeline* de dados para *streaming*, *bacth*, possui fácil monitoramento dos processos, com painel para as analises de inserção dos dados, permitindo assim a visualização do processo de inserção em *real-time* em nosso *data lake*.

Moderniza facilmente o *data lakes* sem codificação manual ou habilidades especiais, e alimenta suas plataformas analíticas com dados contínuos de qualquer fonte.

Como podemos ver a Figura 15 abaixo, é a configuração do Kafka Consumer, ele conecta nos *brokers* e é gerenciado pelo Zookeeper.

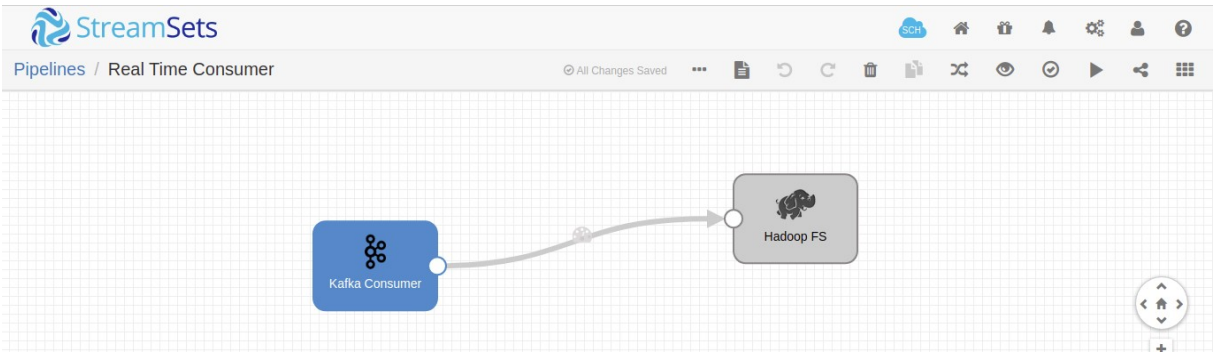

Figura 15 - Kafka *consumer* Streamsets.

Seguindo o fluxo de inserção de dados pelo nosso *pipeline*, temos também a configuração do HDFS, passamos o endereço do HDFS definido no *cluster* Hadoop, no arquivo de configuração, e também definimos o diretório que os dados serão armazenados.

Como veremos nas três figuras a seguir a evolução do processo de inserção de dados no *data lake*, vejamos então o que representa os gráficos, coluna azul representa entrada (*input*) no Kafka Broker, ou seja, significa que o Kafka consumiu os dados do *cluster* Kafka que configuramos, fazendo a leitura dos dados que estão armazenados no tópico que configuramos para receber os *streaming* de dados, a coluna verde, a saída (*output*), representando a gravação dos dados do HDFS, tudo que está a entrar no Kafka Consumer está sendo gravado no HDFS e os (error) são as tentativas da combinação dos processo de *input* e *output* que por algum motivo falharam, e se falhar serão guardados em um arquivo de *logs*.

Como podemos ver na Figura 16 abaixo temos o início do processo de inserção feito no Streamsets.

Figura 16 - Inserção *data lake* no início.

Após uma hora de inserção de dados, conseguimos mais de 40 mil linhas de informações de acessos ao *website*, Figura 17.

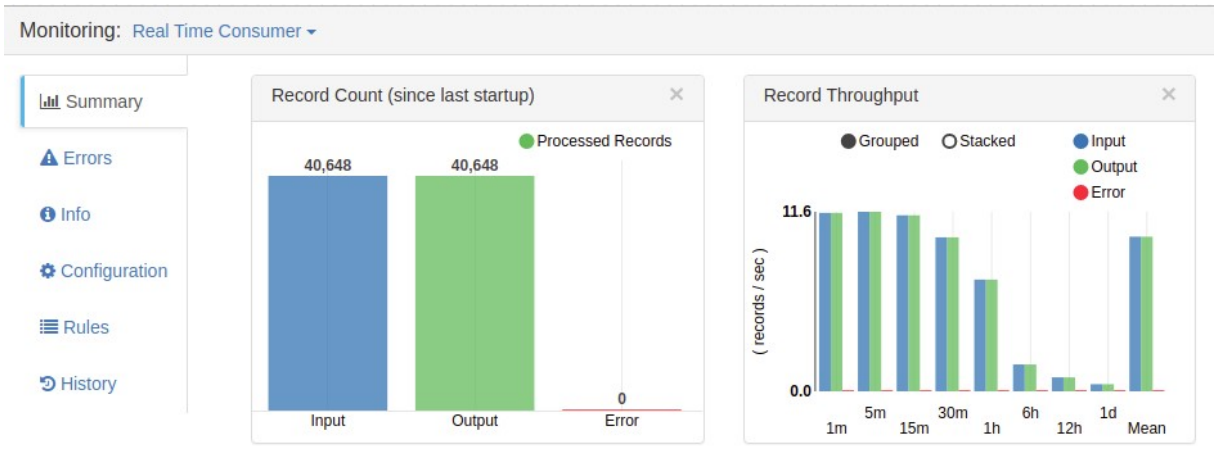

Figura 17 - Inserção 1 hora a correr, painel geral.

Simulamos a inserção de dados durante um dia, fizemos a inserção de 1 milhão linhas de informações de acessos ao *website*, Figura 18.

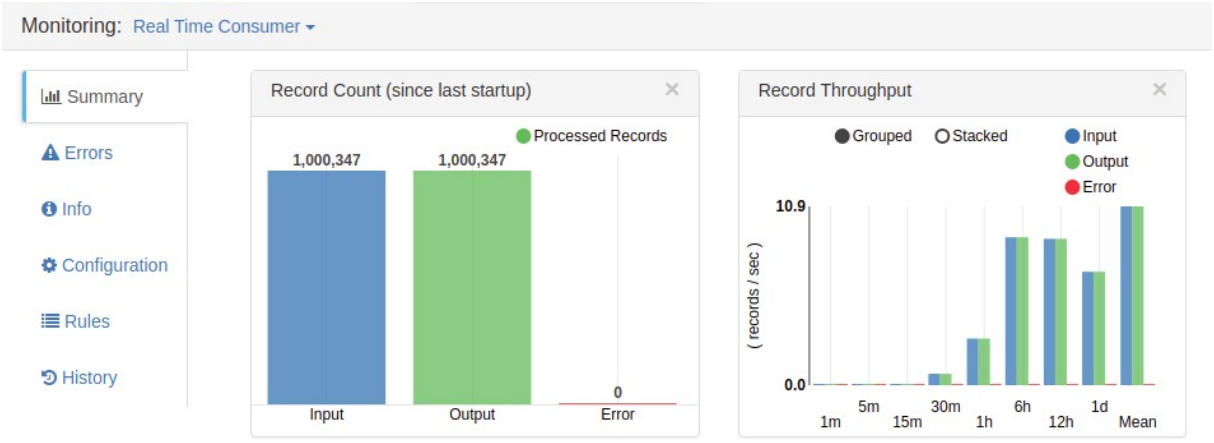

Figura 18 - Inserção 1 dia a correr, painel geral.

Vamos então analisar as três figuras acima, Figura 16, Figura 17 e Figura 18. Comparando o primeiro gráfico, contagem de gravação no HDFS (*Record Count*), podemos ver a evolução das gravações no *data lake* durante o período diário

O segundo gráfico, representa a taxa de gravação durante o período do diário, o eixo do x representa o intervalo de gravação ao decorrer do dia, por exemplo, no minuto 1 e no minuto 5, assim por diante, coluna azul representa entrada (*input*) no Kafka Broker e a coluna verde a saída (*output*).

Nesta Figura 19 fizemos uma listagem do conteúdo inserido em nosso *data lake*, na pasta que configuramos no Streamsets, podemos ver os dados organizados por hora, facilitando assim a posterior leitura dos mesmos.

```
drwxr-xr-x   - hadoop supergroup          0 2021-01-17 18:32 /user
[hadoop@hdpmaster ~]$ hdfs dfs -ls /tmp/out/
Found 27 items
drwxrwxrwx   - root supergroup          0 2021-03-30 23:38 /tmp/out/2021-03-30-22
drwxrwxrwx   - root supergroup          0 2021-03-31 22:55 /tmp/out/2021-03-31-21
drwxrwxrwx   - root supergroup          0 2021-07-05 22:59 /tmp/out/2021-07-05-20
drwxrwxrwx   - root supergroup          0 2021-07-05 23:59 /tmp/out/2021-07-05-21
drwxrwxrwx   - root supergroup          0 2021-07-06 00:59 /tmp/out/2021-07-05-22
```

Figura 19 - Armazenamento no *data lake*, HDFS.

4.5. RECOMENDAÇÕES EM REAL-TIME COM PYSPARK

Além de armazenarmos os dados de *streaming*, também temos que gerar recomendação para cada acesso em *real-time*, para gerar as recomendações desenvolvemos um algoritmo de recomendação que usa factorização de matriz, como falamos no capítulo 2, item 2.2.1, isso nos permitiu gerar as recomendações em *real-time* para cada utilizador do *website*.

O *consumer* por sua vez ira receber os dados de *streaming* e processar.

A Figura 20 abaixo, mostra o código desenvolvido em PySpark que permite abrirmos uma conexão de *streaming* para que conseguíssemos conectar no Kafka Broker. Em destaque o endereço do *broker* 9092.

Figura 40 - Código em PySpark, API do Spark para o Python.

O código acima mencionado, permite ler o *streaming* de dados que estão a entrar no Kafka Broker e está sendo consumido em *real-time* e sendo processado, para que possamos gerar as recomendações em *real-time*, permitindo assim que os utilizadores recebem uma recomendação personalizada de pessoas que acessaram as mesmas coisas, proporcionando assim maior tempo de navegação e maior quantidade de *views*.

A Figura 21 abaixo representa o *job* Spark para que o programa inicie e produza as recomendações personalizadas, o código PySpark é colocado em execução, a sua inicialização vai criar um Spark Streaming que vai fazer a leitura dos dados que estão no Kafka Broker , vamos então fazer uma transformação nos mesmo, essa poderiam ser qualquer modelagem necessária, mas no nosso caso aqui seriam pequenas, pois já recebemos os dados em um formato que permitiu o fácil processamento, possibilitando assim a passagem para o algoritmo de *machine learning* exatamente o código, para que o algoritmo retorne as recomendações baseadas no histórico de dados, proporcionado assim o maior numero de acessos.

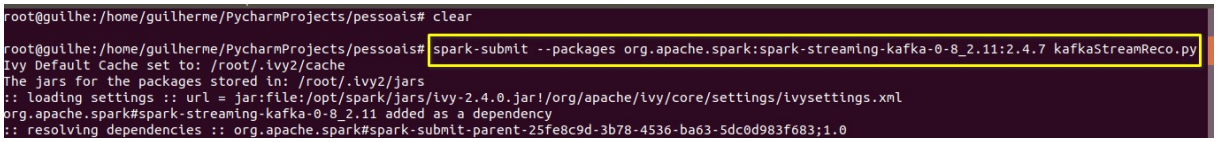

Figura 41 - *Job* Spark.

Abaixo, Figura 22, temos a recomendações impressas no console utilizando o algoritmo mencionado no capítulo 2 item 2.2.1, filtragem colaborativa, estás recomendações serão escritas em uma base de dados, como o MongoDB ou Redis, para que possam ser feitas rápidas leituras pelo *website* com o menor tempo possível de latência, e assim preenchendo o espaço destinado para próximos artigos a serem lidos ou como por exemplo, quem comprou isso também viram esses produtos.

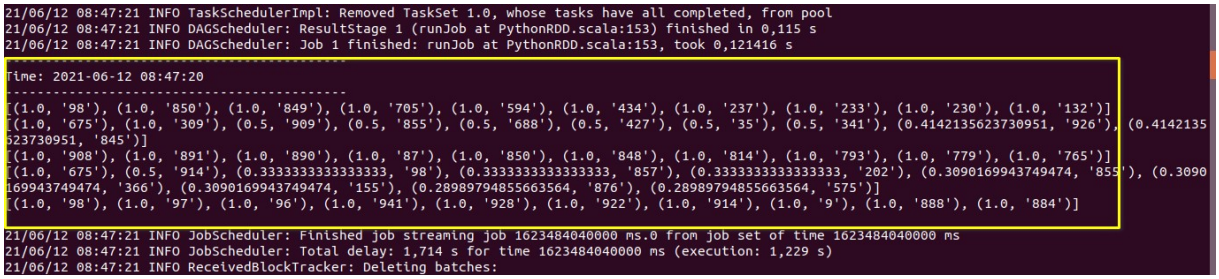

Figura 42 - Recomendações *real-time*.

Como podemos ver na Figura 22, o Spark Streaming está realizando 10 recomendações de outros artigos para a leitura do utilizador ficar mais completa, essas são baseadas nos algoritmos de *machine learning* ALS do MLlib do Spark, que citamos no item 2.2.1 do segundo capítulo deste trabalho. As 10 recomendações foi um número pré-definido no código quando passamos um utilizador e o número de recomendações para o mesmo, isso depende muito do espaço disponível para serem colocadas as recomendações. Como vimos na Figura 11, a Amazon recomendou 5 outros livros quando estávamos a acessar o *site*.

Abaixo, na Tabela 2, podemos ver os artigos que cada um dos utilizadores estava a acessar.

Utilizador	Artigos atuais
1	https://menshealth.pt/sexo/mulheres-mais-bonitas-do-mundo/
20	https://menshealth.pt/fitness/poder-da-prancha/
35	https://www.vdigital.pt/noticias/saudavel/interior/seis-exercicios-para-tonificar-os-gluteos-11193333.html
63	https://www.vdigital.pt/noticias/desporto/interior/lembra-se-deste-golo-historico-de-cristiano-ronaldo-11249256.html
70	https://www.vdigital.pt/programa/nutricao-com-coracao/episodio/pate-de-abacate-e-humus-para-agucar-o-apetite-11213288.html

Tabela 4 – Artigos no momento que as recomendações foram feitas.

As 10 recomendações são ordenadas por peso, de forma decrescente, sendo o maior peso a primeira da lista, isso não importa muito, vejamos, temos 10 espaços para colocar as recomendações, alguns são destinados para publicidades, a intensão é que o utilizador veja os 10 artigos, todavia, temos que fazer uma futura analise de cliques por espaço, mapa de calor, testes A/B, dentre outros métodos, para podermos avaliar a disposição dos artigos perante os espaços disponíveis.

Como mencionamos acima, os pesos são calculados e ordenados em ordem decrescente, a seguir segue uma breve explicação do cálculo dos pesos, assim como já vimos no capítulo 2, item 2.2.1. Vejamos então como calculamos os pesos, denominamos artigo x e artigo y, para facilitar a compreensão, o artigo y é acessado 2 vezes depois do artigo x, por um utilizador, isso se repete por milhões de vezes, resultando em uma matriz *user*X*webpage*, onde fazemos a multiplicação da matriz em função de *user*, por a matriz em função de *webpage*.

Para ficar claro como o algoritmo funciona, vamos descrever os artigos recomendados para cada um dos 3 primeiros utilizadores, seguindo a ordem impressa no console, Figura 22.

4.5.1. Utilizador 1

- Artigo atual;
  https://menshealth.pt/sexo/mulheres-mais-bonitas-do-mundo/

ID_Artigo	Peso	Artigos recomendados
98	1.0	https://menshealth.pt/sexo/mulheres-mais-bonitas-do-mundo/

580	1.0	https://www.vdigital.pt/noticias/desporto/interior/a-reacao-do-slavia-praga-ao-saber-que-ficou-no-grupo-da-morte-11252355.html
849	1.0	http://menshealth.pt/sexo/quais-as-medidas-do-homem-perfeito
785	1.0	https://menshealth.pt/fitness/qual-tempo-maximo-ficar-prancha/
594	1.0	https://menshealth.pt/perder-peso/estrategias-eficazes-perder-barriga/
434	1.0	https://menshealth.pt/sexo/ginasios-amentam-infidelidade/
237	1.0	https://www.vdigital.pt/noticias/mundo/interior/autoridades-portuguesas-salvam-mais-de-100-migrantes-em-dois-dias-11252757.html
233	1.0	https://www.vdigital.pt/noticias/mundo/interior/perseguicao-policial-a-nado-na-praia-de-barcelona-11251989.html
230	1.0	http://menshealth.pt/nutricao/grelhado-receita-frango-picante
132	1.0	https://menshealth.pt/novidades/mh-setembro-especial-desporto/

Tabela 3 - Recomendações para utilizador 1

4.5.2. Utilizador 20

- Artigo atual;
  https://menshealth.pt/fitness/poder-da-prancha/

ID_Artigo	Peso	Artigos recomendados
675	1.0	https://www.vdigital.pt/noticias/desporto/interior/um-golo-sem-fair-play-que-obrigou-jogador-a-fugir-do-relvado-11252287.html
309	1.0	https://menshealth.pt/novidades/fotos-nuas-paisagens-tendencia/
909	0.5	https://www.vdigital.pt/noticias/desporto/interior/a-celebracao-do-estrela-vermelha-a-bordo-de-um-taque-de-guerra-11251968.html
955	0.5	https://menshealth.pt/fitness/5-passos-um-six-pack-2018/
688	0.5	https://www.vdigital.pt/noticias/lifestyle/interior/parece-magia-o-novo-bmw-vbx6-confunde-o-cerebro-11253438.html
427	0.5	http://menshealth.pt/saude/oms-dizer-cigarros-eletronicos
35	0.5	https://www.vdigital.pt/noticias/desporto/interior/a-danca-de-jorge-fonseca-depois-de-se-tornar-campeao-do-mundo-de-judo-em--100-kg-11252908.html
341	0.5	https://menshealth.pt/
926	0.41	http://menshealth.pt/novidades/carros-proximo-james-bond-007
845	0.41	https://www.vdigital.pt/noticias/desporto/interior/a-atitude-grandiosa-de-lisandro-lopez-apos-lesao-grave-de-adversario-11248866.html

Tabela 4 - Recomendações para utilizador 20

4.5.3. Utilizador 35

- Artigo atual;
  https://www.vdigital.pt/noticias/saudavel/interior/seis-exercicios-para-tonificar-os-gluteos-11193333.html

ID_Artigo	Peso	Artigo recomendados
908	1.0	http://menshealth.pt/perder-peso/dieta-vida-nao-cometa-erros
891	1.0	https://www.vdigital.pt/noticias/numeros/interior/as-despesas-que-podem-estar-a-boicotar-as-suas-poupancas-11252163.html
890	1.0	https://www.vdigital.pt/noticias/lifestyle/interior/como-visitar-uma-destilaria-de-dublin-sem-sair-de-lisboa-11252129.html
87	1.0	https://menshealth.pt/saude/ate-idade-cresce-penis/
850	1.0	https://menshealth.pt/sexo/5-dicas-para-melhor-sexo-oral-a-opiniao-delas/
848	1.0	https://menshealth.pt/sexo/45-posicoes/
814	1.0	https://menshealth.pt/nutricao/truque-dos-espanhois-para-perder-peso/
793	1.0	http://menshealth.pt/estrategia/deve-vestir-vermelho-no-ginasio
779	1.0	https://menshealth.pt/novidades/mens-health-julho-agosto/
765	1.0	https://www.vdigital.pt/noticias/numeros/interior/siga-a-viagem-das-notas-ate-chegarem-a-sua-carteira-11246399.html

Tabela 5 - Recomendações para utilizador 35

Como podemos ver nas tabelas 3, 4 e 5, as recomendações fazem sentido, podendo melhorar, mas o projeto como um todo, tem muito mais estrutura do que apenas *machine learning*.

Os modelos de fator latente mais bem-sucedidos são baseados em MF. Em sua forma básica, o MF caracteriza itens e utilizadores por vetores ou fatores inferidos dos padrões de classificação dos itens. A alta correspondência entre os fatores do utilizador e o fator do item leva a uma recomendação. Esses métodos se tornaram populares recentemente, combinando boa escalabilidade com precisão preditiva. Eles oferecem maior flexibilidade para modelar várias aplicações da vida real.

5. CONCLUSÕES

Considerando a complexidade do projeto como um todo, desde a definição do problema de negócio até a recomendação em final, tivemos o objeto concluído, como muitas limitações é claro, mas vejo todas a partes em funcionamento. O algoritmo de *machine learling* não foi o autor principal nesse projeto, construímos um processo básico, porém ele executa as recomendações, sendo assim conseguimos ver todo *pipeline* em atividade.

O processo de configuração das máquinas e ferramentas custou a maior parte do tempo, com resultado disso tudo o algoritmo funciona muito bem, melhor ainda a integração dos serviços *open sources* usados para a composição do projeto. O Kafka faz o melhor trabalho, permitindo que acessarmos os dados em *streaming*, o Spark, responsável pelo processamento em ambiente distribuído, para treinar o algoritmo com os dados que estão armazenados no HDFS e fazer as previsões com o Spark Streaming, o HDFS que vez faz parte do ecossistema Hadoop, onde os dados são armazenados de forma distribuída permitindo que o Spark processe esses dados também de forma distribuída, já que ambos nasceram na era do *big data* e estão muito bem preparados para esse novo conceito que vivemos nos dias atuais.

O *cluster* Kafka está configurado com três *brokers*, como vimos na Figura 3, isso significa que a qualquer momento poderemos adicionar mais *brokers* ao *cluster* Kafka, nos permitindo a acompanhamento do crescimento das entradas de mensagens. O *cluster* Hadoop segue a mesma configuração do *cluster* Kafka, é facilmente escalonável, a configuração inicial do *cluster* Hadoop temos 16 GB de memória RAM, e um espaço em disco totalmente configurável.

O Spark está confortável para processar milhões de registros nessa configuração, mas claro, podendo ser facilmente alterado para consegui processar mais, e acompanhar o crescimento do projeto.

As recomendações por sua vez, são gerados no final do *pipeline* e como podemos ver no capítulo 4, item 4.5, os utilizadores que estava a acessar o *website* naquele momento, receberam mais recomendações similares ao conteúdo que estavam a ver.

Estou muito satisfeito com o projeto como um todo, as recomendações são apenas a parte final no processo todo, e farão parte da leitura do utilizador proporcionando maior números de visitas ao *website*.

6. LIMITAÇÕES E RECOMENDAÇÕES PARA TRABALHOS FUTUROS

6.1. LIMITAÇÕES

O maior problema foi o equipamento, com as limitações de processamento precisamos deixar o armazenamento no HDFS para ser executado com outra VM (do inglês, *Virtual Machine*), permitindo que o Spark e o *cluster* Kafka com 3 *brokers* funcionassem muito melhor em outra VM.

Na parte do cliente, tivemos muito burocracia, falta de conhecimento técnico por parte dos gestores, falta de recurso financeiro para o investimento no negócio, tornando assim a implementação quase impossível, a maior parte do projeto, melhor todo o projeto, foi homologado no *desktop* que providenciei por conta própria.

6.2. RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Uma recomendação para trabalhos futuros é implementação de uma funcionalidade para armazenar o resultado das recomendações em um banco de dados não relacional, como o MongoDB ou Redis.

Também recomendamos de fazer o *deploy* com o Docker (Merkel, 2014), que permitem a implantação de novas versões em paralelo e independentemente de outros serviços. Os desenvolvedores podem trabalhar em paralelo e obter alterações na produção de forma independente e mais rápida. Permite a entrega e implantação contínuas de aplicativos grandes e complexos de modelos *machine learning*. Se houver preocupações com a privacidade de dados na implantação de modelos de AI na nuvem, a criação de modelos individuais como *containers* permitirá a implantação de modelos de AI no ambiente local. Na maioria dos projetos de AI, haverá vários modelos de AI que serão desenvolvidos para executar funções específicas (por exemplo: um modelo para reconhecimento de imagens, modelo para extração de informações, etc.). O Docker permite que esses modelos sejam desenvolvidos, atualizados e implantados de forma independente.

Fazer a orquestração dos *containers* Docker com o Kubernets (Burns, Beda e Hightower, 2019)

Em arquiteturas de microsserviço, os aplicativos são divididos em vários serviços distintos, cada um empacotado em um *container* separado. O benefício, especialmente para organizações que aderem às práticas de integração contínua e entrega contínua (CI/CD) (do inglês, *Continuous Integration/Continuous Delivery*), é que os *containers* são escalonáveis e efêmeros - instâncias de aplicativos ou serviços hospedados em *containers* vêm e vão conforme a necessidade.

7. BIBLIOGRAFIA

Adomavicius, Gediminas & Tuzhilin, Alexander. (2005) - Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering. ISSN 10414347. 17:6 734–749. doi: 10.1109/TKDE.2005.99.

Alpaydin, Ethem. (2020) - Introduction to Machine Learning. Massachusetts Institute of Technology. 4º Edition

Bhole, Preeti; Chandrakar, Neetish Kumar & Swarnkar, Virendra. (2017) - Review of Recommendation System for Web Application. International Journal of Science and Research (IJSR). 6:1 314–317. doi: 10.21275/art20163898.

Bokde, Dheeraj; Girase, Sheetal & Mukhopadhyay, Debajyoti. (2015) - Matrix Factorization model in Collaborative Filtering algorithms: A survey. Procedia Computer Science. ISSN 18770509. 49:1 136–146. doi: 10.1016/j.procs.2015.04.237.

Borthakur, Dhruba. (2007) - The hadoop distributed file system: Architecture and design. Hadoop Project Website. 1–14.

Borthakur, Dhruba. (2008) - HDFS architecture guide. The Apache Software Foundation. 1–14.

Burns, Brendan; Beda, Joe & Hightower, Kelsey. (2019) - Kubernetes up and running. ISBN 978-1-492-04653-0.

Cazella, Sílvio César; Reategui, Eliseo Berni; Machado, Munique & Barbos, Jorge Luis V. (2009) - Recomendação de Objetos de Aprendizagem Empregando Filtragem Colaborativa e Competências. Anais do Simpósio Brasileiro de Informática na Educação. 1:1. doi: http://dx.doi.org/10.5753/cbie.sbie.2009.%25p.

Brink, Henrik; Richards, Joseph W. & Fetherolf, Mark. (2017) - Real-World Machine Learning. Manning Publications Co. ISBN 9781617291920.

Cruz, Joseph A. & Wishart, David S. (2006) - Applications of machine learning in cancer prediction and prognosis. Departments of Biological Science and Computing Science. ISSN 11769351. 2:1 59–77. doi: 10.1177/117693510600200030.

Sobral, Bosco. (2011) - Um Exemplo Didático de Aplicação Hadoop. www.inf.ufsc.br - Unidade V 5. 53–62.

Drakonaki, Eleni E. & Allen, Gina M. (2010) - Magnetic resonance imaging, ultrasound and real-time ultrasound elastography of the thigh muscles in congenital muscle dystrophy. Skeletal Radiology. ISSN 03642348. 39:4 391–396. doi: 10.1007/s00256-009-0861-0.

McCarthy, John. (2004) - What Is Artificial Intelligence? Computer Science Department Stanford University. 1–14. doi: 10.7551/mitpress/12518.003.0004.

White, Tom. (2012) - Hadoop: The Definitive Guide. O'Reilly. ISBN 9781449389734.

Gandhi, Mrunal; Nikam, Tushar; Shahane, Akshay; Shete, Mayur & Dr. Nalavade, Kamini C. - Product Recommendation System Using Machine Learning Based Collaborative Filtering Technique. International Journal of Science and Research (IJSR). 9:12 (2020) 176–180. doi: 10.21275/SR201127214109.

Gao, Jim & Jamidar, Ratnesh. (2014) - Machine Learning Applications for Data Center Optimization. Google White Paper. 1–13.

Giebler, Corinna; Gröger, Christoph; Hoos, Eva; Eichler, Rebecca; Schwarz, Holger & Mitschang, Bernhard. (2021) - The Data Lake Architecture Framework: A Foundation for Building a Comprehensive Data Lake Architecture. Proceedings der 19. Fachtagung Datenbanksysteme für Business, Technologie und Web.

Hitzler, Pascal & Janowicz, Krzysztof. (2019) - The Semantic Web. 4th Paradigm Editorial. Springer

Karau, Holden; Konwinski, Andy; Wendell, Patrick &. Zaharia, Matei. (2015)- Learning Spark: Lightning-Fast Big Data Analysis. Databricks.

Inmon, William H. (1995) - What is a data warehouse? Prism Tech Topic 1:1 1–12.

Jena, Bibhudutta; Gourisaria, Mahendra Kumar; Rautaray, Siddharth Swarup & Pandey, Manjusha. (2017) - A survey work on optimization techniques utilizing map reduce framework in HADOOP cluster. International Journal of Intelligent Systems and Applications. ISSN 20749058. 9:4 61–68. doi: 10.5815/ijisa.

Kross, Johannes & Krcmar, Helmut. (2016) - Modeling and Simulating Apache Spark Streaming Applications. Softwaretechnik-Trends. 36:4 3–5.

Kulkarni, Amogh Pramod & Khandewal, Mahesh. (2014) - Survey on Hadoop and Introduction to YARN. International Journal of Emerging Technology and Advanced Engineering. 4:5 82–87.

Lachi, Ricardo Luís & Vieira Da Rocha, Heloísa. (2005) - Aspectos básicos de clustering: conceitos e técnicas. Núcleo de Informática Aplicada à Educação (Nied), UNICAMP - Instituto de Computação – Universidade Estadual de Campinas.

Langley, Pat. (2011) - The changing science of machine learning. Machine Learning. Springer. ISSN 08856125. 82:3 275–279. doi: 10.1007/s10994-011-5242-y.

Li, Yamin & Peng, Shietung. (2000) - Dual-cubes: a new interconnection network for high-performance computer clusters. Proceedings of the 2000 International Computer Symposium, Workshop on Computer Architecture. 51–57.

Machine learning: o que é e qual sua importância? | SAS - [Em linha], atual. 2021. [Consult. 20 jul. 2020]. Disponível em https://www.sas.com/pt_br/insights/analytics/machine-learning.html.

Mehta, Sneha & Mehta, Viral. (2016) - Hadoop Ecosystem: An Introduction. International Journal of Science and Research (IJSR). 5:6 557–562. doi: 10.21275/v5i6.nov164121.

Merkel, Dirk. (2014) - Docker : Lightweight Linux Containers for Consistent Development and Deployment Docker : a Little Background Under the Hood. Linux Journal. 2014:239 2–7.

Meteren, Robin Van & Someren, Maarten Van. (2000) - Using Content-Based Filtering for Recommendation. ECML/MLNET Workshop on Machine Learning and the New Information Age. ISSN 15506606. 47–56.

Milone, D. H.; Azar, A. & Rufiner, H. L. (2002) - Supercomputadoras basadas en " clusters " de PCs. Center for Signals, Systems and Computational Intelligence.

O'malley, Owen. (2008) - Terabyte sort on apache hadoop. Yahoo. May 1–3.

Paas, S.; Dormanns, M.; Bemmerl, T.; Scholtyssik, Karsten & Lankes, S. (1997)- Computing on a

Cluster of PCs : Project Overview and Early Experiences. http://www.lfbs.rwth-aachen.de/.

Pasquinelli, Matteo. (2009) - Google's PageRank Algorithm: A diagram of cognitive capitalism and the rentier of the common intellect. Deep Search: The Politics of Search Beyond Google. May 2012 152–162.

Romero, C.; Oliveira, H. P. (2011) - Exact solutions in brans-dicke theory: A dynamical system approach. Astrophysics and Space Science. ISSN 0004640X. 159:1 1–9. doi: 10.1007/BF00640482.

Rosindell, James; Wong, Yan. (2018) - Biodiversity, the tree of life, and science communication. Phylogenetic Diversity: Applications and Challenges in Biodiversity Science. Phylogenetic diversity. 2 41–71. doi: 10.1007/978-3-319-93145-6_3.

Sathyapriya, M. & Thiagarasu, V. (2015) - Big Data Analytics Techniques for Credit Card Fraud Detection: A Review. International Journal of Science and Research. 6:5 2319–7064.

Schafer, J.B., Konstan, J.A. & Riedl, J. (2001) - E-Commerce Recommendation Applications. Data Mining and Knowledge Discovery 5, 115–153. https://doi.org/10.1023/A:1009804230409

Schema-on-Read vs Schema-on-Write - Blog | luminousmen - [Em linha], atual. 2020. [Consult. 10 nov. 2020]. Disponível em https://luminousmen.com/post/schema-on-read-vs-schema-on-write.

Sharma, Disha & Kaur, Sumit. (2016) - A Robust Multi-Factor Recommender System for Online Libraries and E-book Portals. International Journal of Science and Research (IJSR). 5:6 37–39. doi: 10.21275/v5i6.nov163421.

Shinde, Santosh A. & Pradip N. Shendage. (2015) - A Novel QoS-Based Approach for Web Service Recommendation and Visualization. International Journal of Science and Research (IJSR). ISSN 2319-7064. 4:6 332–336.

Sigtia, Siddharth; Boulanger-Lewandowski, Nicolas & Dixon, Simon. (2015) - Audio Chord Recognition with a Hybrid Recurrent Neural Network. ISMIR. Section 3 127–133.

Singh, Ajit & Ahmad, Sultan. (2019) - Architecture of Data Lake. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. March 411–414. doi: 10.32628/cseit1952121.

Spark Streaming | Apache Spark - [Em linha], atual. 2018. [Consult. 10 nov. 2020]. Disponível em https://spark.apache.org/streaming/.

Trindade, Lucas Lopes. (2018) - Detecção de Obstáculos para Carros Autônomos utilizando Aprendizado Profundo. Universidade Federal de Santa Catarina Departamento de Engenharia Elétrica e Eletrônica.

Weiss, E. A. (2002) - Biographies: Eloge: Arthur Lee Samuel (1901-90). IEEE Annals of the History of Computing. . ISSN 1058-6180. 14:3 (2002) 55–69. doi: 10.1109/85.150082.

What is open source software? | Opensource.com - [Em linha], atual. 2021. [Consult. 10 nov. 2020]. Disponível em https://opensource.com/resources/what-open-source.

Zaharia, Matei; Chowdhury, Mosharaf; Das, Tathagata; Dave, Ankur; Ma, Justin; Mccauley, Murphy; Franklin, Michael J.; Shenker, ScotT & Stoica, Ion. (2012) - Fast and Interactive Analytics over Hadoop Data with Spark. Networked Systems usenix.org. 37:4 45–51.

Zaki, Nashwan Dheyaa, Hashim Nada, Yousif; Mohialden, Yasmin Makki; Mohammed, Mostafa Abdulghafoor; Sutikno, Tole & Ali, Ahmed Hussein. (2020) - A real-time big data sentiment analysis for iraqi tweets using spark streaming. Bulletin of Electrical Engineering and Informatics. . ISSN 23029285. 9:4 1411–1419. doi: 10.11591/eei.v9i4.1897.

Zhang, Shuai; Yao, Lina; Tay, Yi; Xu, Xiwei; Zhang, Xiang & Zhu, Liming. (2018) - Metric Factorization: Recommendation beyond Matrix Factorization. School of Computer Science and Engineering.